Теоретические вопросы программирования

Министерство народного образования Латвийской ССР
Латвийский ордена Трудового Красного Знамени
государственный университет имени П.Стучки
Вычислительный центр

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ ПРОГРАММИРОВАНИЯ

СБОРНИК НАУЧНЫХ ТРУДОВ

Латвийский государственный университет им.П.Стучки
Рига 1988

УДК 519.95

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ ПРОГРАММИРОВАНИЯ

Теоретические вопросы программирования: Сборник научных трудов /Отв.ред. Р.В.Фрейвалд. Рига: ЛГУ им.П.Стучки, 1988. - 127 с.

Сборник посвящен исследованию сложности различных типов вычислительных устройств, таких как альтернирующие и вероятностные машины, а также проблемам индуктивного синтеза программ.

Сборник рассчитан на научных работников, занимающихся или интересующихся теорией алгоритмов, аспирантов и студентов.

Библ.назв.- 71 назв.

Печатается по решению Издательского совета ЛГУ
им.П.Стучки

Т 30502-146у
М812(11)-88 19.88.2405000000

ВВЕДЕНИЕ

Теоретической базой работ по развитию математического обеспечения ЭВМ служит теория алгоритмов и программ. В последнее время созданы удивительные по своей силе алгоритмы вычисления простейших функций, основанные на глубоких результатах классической математики. Поэтому не случайно, что как в нашей стране, так и за рубежом ведутся интенсивные исследования на стыках теории программирования и классических дисциплин математики.

Настоящий сборник посвящен различным вопросам теории алгоритмов. Часть работ относится к тематике сложности вычислений. В частности, получены важные результаты, характеризующие сложностные преимущества вероятностных алгоритмов перед детерминированными.

В сборнике рассмотрены и более нетрадиционные разделы теории алгоритмов, например, характеризация детерминированных и недетерминированных алгоритмов, которые никогда не останавливаются (как операционные системы ЭВМ).

Большая часть сборника посвящена такому совершенно новому направлению, как автоматический синтез программ. Здесь впервые с полным доказательством излагаются результаты, полученные в ВЦ при ЛГУ им. П. Стучки и получившие широкую известность как в нашей стране, так и за рубежом.

ИНДУКТИВНЫЙ СИНТАКСИЧЕСКИЙ СИНТЕЗ ПРОГРАММ ПО НЕПОЛНЫМ ПРИМЕРАМ

Е.Б.Кинбер
ВЦ при ЛГУ им.П.Стучки

Язык многоточечных выражений, разработанный Я.М.Барздинем [1,2] позволяет осуществлять индуктивный синтез по примерам вычислений многих алгоритмов, не содержащих сложных ветвлений. Однако для большого числа таких алгоритмов полный пример вычисления является очень длинным; в то же время пример естественным образом можно разделить на фрагменты, которые по существу не отличаются друг от друга последовательностью и характером выполняемых алгоритмом действий (точнее, параметры этих фрагментов отличаются не более чем на один шаг арифметической прогрессии). Алгоритмами такого типа являются, в частности, многие алгоритмы с рекурсией. Для алгоритмов такого рода вполне естественным представляется в качестве исходного материала для синтеза подавать лишь один фрагмент примера, вполне характеризующий алгоритм упомянутым выше образом, а также информировать синтезатор, что "далее алгоритм работает аналогично". В данной работе предлагается некоторый язык, удобный, с одной стороны, для записи большого ряда алгоритмов и, с другой стороны, позволяющий достаточно эффективно синтезировать эти алгоритмы по входной информации упомянутого выше типа. Язык объединяет в себе традиционные средства языков программирования (например, присваивания, WHILE-операторы) со средствами, удобными для синтаксического синтеза, когда входная информация рассматривается как чисто символьная строка без учета семантики. В качестве последних фигурируют, например, звездочка Клини (для циклов) и объединение $\cup$ (для условных операторов), впервые введенные в [3,4] при разработке модели синтаксического индуктивного синтеза.

Основное внимание в работе уделяется демонстрации практических возможностей языка и синтезатора, иногда в

ущерб точности изложения.

I. ПРИМЕРЫ ВЫЧИСЛЕНИЙ

В качестве первого примера рассмотрим алгоритм приведения матрицы с рациональными элементами к диагональному виду - основную часть алгоритма Гаусса для решения систем линейных уравнений. Имеется ввиду матрица размера $m \times n$ $(m \geq n)$, элементами которой являются $a(i,j)$, $1 \leq i \leq m$, $1 \leq j \leq n$. Пример вычисления снабжается разъясняющими аннотациями, заключенными в скобки [,].

ПРИМЕР I. Приведение матрицы к диагональному виду. Алгоритм Гаусса приведения матрицы к диагональному виду можно разъяснить с помощью следующего примера

INPUT: a ч ч а у $a(i,j)$, $1 \leq i \leq 4\,[=m]$,

$1 \leq j \leq 5\,[=n]$;

$i := 2$, $j := 1\,[= \underline{i-1}]$;

$$a(2[=i], 1[=j]) := \frac{a(2[=i], 1[=j])}{a(2[=i], 1[=\underline{i-1}])} \cdot a(1[=\underline{i-1}], 1[=\underline{i-1}]) - a(1[=\underline{i-1}], 1[=j]);$$

$$a(2,2) := \frac{a(2,2)}{a(2,1)} \cdot a(1,1) - a(1,2);$$

$$a(2,3) := \frac{a(2,3)}{a(2,1)} \cdot a(1,1) - a(1,3);$$

$$a(2,4) := \frac{a(2,4)}{a(2,1)} \cdot a(1,1) - a(1,4);$$

$$a(2,5[=n]) := \frac{a(2,5[=n])}{a(2,1)} \cdot a(1,1) - a(1,5[=n]);$$

$i := 3$; $j := 2\,[= \underline{i-1}]$; P;

Если $i = 4[=m]$, то $STOP$.

Символ P означает здесь и ниже: "повторить те же действия, что и выше для новых значений переменных".

Рассмотрим теперь более сложный пример - алгоритм

сортировки посредством выделения максимального элемента и удаления его из массива.

ПРИМЕР 2. Сортировка-I. Дан массив $K(1), K(2), \ldots, K(m)$, который требуется переслать в массив $M(1), M(2), \ldots, M(m)$ в отсортированном виде. Пример, представляющий алгоритм, можно записать следующим образом:

INPUT: аччау $K(1), K(2), \ldots, K(12[=m])$; $y:=12[=m]$; $x:=1$;

$K(1[=x]) \leqslant K(2[=x+1])$?; допустим да;

$K(2[=x]) \leqslant K(3[=x+1])$?; допустим да;

$K(3[=x]) \leqslant K(4[=x+1])$?; допустим да;

$K(4[=x]) \leqslant K(5[=x+1])$?; допустим нет;

$K(4[=x]) \longleftrightarrow K(5[=x])$;

$K(5[=x]) \leqslant K(6[=x+1])$?; допустим да;

$K(6[=x]) \leqslant K(7[=x+1])$?; допустим да;

$K(7[=x]) \leqslant K(8[=x+1])$?; допустим нет;

$K(7[=x]) \longleftrightarrow K(8[=x+1])$;

$K(8[=x]) \leqslant K(9[=x+1])$?; допустим да;

$K(9[=x]) \leqslant K(10[=x+1])$?; допустим да;

$K(10[=x]) \leqslant K(11[=x+1])$?; допустим да;

$K(11[=x]) \leqslant K(12[=x+1=y])$?; допустим нет;

$K(11[=x]) \longleftrightarrow K(12[=x+1=y])$;

$M(12[=y]) \longleftrightarrow K(12[=y])$;

$y:=11$; $x:=1$; P;

Если $y=1$, то $M(1) \leftarrow K(1)$; STOP.

Рассмотрим, наконец, пример вычисления алгоритма типа "разделяй и властвуй", использующего механизм рекурсии в полной мере. В качестве такого алгоритма мы выберем известный алгоритм "быстрой сортировки".

ПРИМЕР 3. Сортировка-2. Требуется упорядочить массив $K(1), K(2), \ldots, K(m)$. Пример, задающий алгоритм, можно представить следующим образом.

INPUT: аччау $K(1), K(2), \ldots, K(16[=m])$;

$x_0:=1$; $y_0:=16[=m]$; $x:=1[=x_0]$; $y:=16[=y_0]$;

$K(1[=x]) \leqslant K(16[=y])$?; допустим да;

$K(1[=x]) \leqslant K(15[=y])$?; допустим да;

$K(1[=x]) \leq K(14[=y])$?;допустим да;

$K(11[=x]) \leq K(13[=y])$?; допустим да;

$K(1[=x]) \leq K(12[=y])$?; допустим нет;

$K(1[=x]) \longleftrightarrow K(12[=y])$;

$K(2[=x]) \leq K(12[=y])$?; допустим да;

$K(3[=x]) \leq K(12[=y])$?; допустим нет;

$K(3[=x]) \longleftrightarrow K(12[=y])$;

$K(3[=x]) \leq K(11[=y])$?; допустим да;

$K(3[=x]) \leq K(10[=y])$?; допустим да;

$K(3[=x]) \leq K(9[=y])$?; допустим да;

$K(3[=x]) \leq K(8[=y])$?; допустим нет;

$K(3[=x]) \longleftrightarrow K(8[=y])$;

$K(4[=x]) \leq K(8[=y])$?; допустим да;

$K(5[=x]) \leq K(8[=y])$?; допустим да;

$K(7[=x]) \leq K(8[=y])$?; допустим да;

$K(7[=x]) \leq K(8[=y=\underline{x+1}])$?; допустим нет;

$K(7[=x]) \longleftrightarrow K(8[=y=\underline{x+1}])$;

СТЕК:= $(x+1, y_0)$; $y_0 := x-1$; $x := x_0$; $y := y_0$; P;

Если $x_0 = y_0$, то

WHILE (¬Emp(СТЕК)) DO ((x_0, y_0) := СТЕК)); P.

Этот пример вычисления по существу представляет собой линейную запись разъяснения алгоритма с помощью меток [i] и [j] в [5].

В примере фигурируют переменная "СТЕК" и предикат Emp(СТЕК), проверяющий "СТЕК" на пустоту.

ЗАМЕЧАНИЕ. Алгоритм "быстрой" сортировки мы рассматриваем в упрощенной трактовке: в стек засылаются границы не "меньшего", а "правого" из полученных двух массивов. Это позволяет выделить рекурсивный характер алгоритма, избавляясь от вычисления сложных предикатов, связанных с алгебраическими операциями над переменными.

В следующем разделе предлагается язык программирования, удобный для индуктивной формализации примеров вычислений, содержащих символ P (с сопровождающей его семантикой). При этом синтез осуществляется чисто синтаксическими средствами – входная строка рассматривается как после-

довательность символов без учета семантики.

2. ОПРЕДЕЛЕНИЕ КЛАССА ПРОГРАММ

Пусть N – множество всех натуральных, а Z – множество всех целых чисел. Пусть $A = A' \cup N$ – алфавит, где A' – конечное множество символов; A' содержит специальные символы: $:=$ (присваивание) и $=$ (равенство). Пусть X – конечное множество переменных (возможно, $A \cap X = \varnothing$) и СТЕК – специальная стековая переменная. Пусть ОР включает в себя символы $+, -, (,)$, $^{\pm}$ (звездочка Клини) $\pm, =$, WHILE, DO, $\textcircled{\neq}$ (предикатный символ), $\cup$ (объединение); пусть $OP \cap (A \cup X) = \varnothing$.

Далее особую роль играет подалфавит A; в нем записываются примеры вычислений программ.

Областью значений переменных $x \in X$ является N; будем считать, что в X имеются индексированные переменные с индексами из N. Для любой цепочки Q из данного алфавита $X(Q)$ обозначает множество переменных из X, входящих в Q. Выражения $x := x+1$ и $x := x \dot{-} 1$ $(x \in X)$ ниже для краткости будем записывать в виде x^+ и, соответственно, x^-.

Каждая программа представляет собой цепочку $A_1 B A_2$, где A_1, A_2 – слова в алфавите A, а B – выражение вида

$$(2.1)\quad C_1\, WHILE\, (x \textcircled{\neq} \mu)\, DO\, (C_2\, WHILE\, (y \textcircled{\neq} \nu)\, DO\, (Q)\, Q_1\, D),$$

где $x, y \in X(QQ_1D)$, $\mu, \nu \in N \cup X$, если $\mu \in X$, то $\mu \in X(Q_1 D)$, если $\nu \in X$, то $\nu \in X(Q)$, C_1 содержит выражение $x := a$, $a \in N \cup X \cup \{СТЕК\}$ и если $\mu \in X$, то также $\mu := b$, $b \in N \cup X \cup \{СТЕК\}$, C_2 содержит выражение $y := a$, $a \in N \cup X \cup \{СТЕК\}$ и если $\nu \in X$, то также $\nu := b$, $b \in N \cup X \cup \{СТЕК\}$; Q_1 – слово в алфавите $A \cup X$, $D \in \{x^+, x^-, \mu^+, \mu^-, x^+\mu^-, x^-\mu^+\}$; выражение Q определяется следующим образом.

Пусть $\widetilde{X}$ - множество слов вида z, $\underline{z+c}$, $\underline{z-c}$, $z \in X, c \in N$. Тогда Q есть либо слово

(2.2) $T_1 M_1 \cup T_2 M_2 \cup \dots \cup T_m M_m$,

либо

(2.3) $R_0 (T_1 M_1)^* R_1 K_1 (T_2 M_2)^* R_2 K_2 \dots (T_m M_m)^* R_m K_m$,

где

(а) T_i, $1 \le i \le m$, R_i, $0 \le i \le m$, - цепочки в алфавите $A \cup \widetilde{X}$, $\exists i (R_i \neq \Lambda)$;

(б) M_i, $1 \le i \le m$, K_i, $1 \le i \le m$, - цепочки вида $x_1^{a_1} x_2^{a_2} \dots x_p^{a_p}$, где все x_i, $1 \le i \le p$, попарно различны, $a_1, \dots, a_m \in \{+, -\}$, $x \in \{x_1, \dots, x_m\}$, $y \in \{x_1, \dots, x_m\}$ по крайней мере для одного M_i или K_i;

(в) если $z^+ \in M_1 M_2 \dots M_m K_1 K_2 \dots K_m$ то

$z^- \notin M_1 M_2 \dots M_m K_1 K_2 \dots K_m$ и наоборот;
если $y \in \{x_1, \dots, x_m\}$ для одного из M_i или K_i, то

$y^+ \in M_1 \dots M_m K_1 \dots K_m \leftrightarrow y^- \in M_1 \dots M_m K_1 \dots K_m$
и наоборот. Будем считать также, что A_2 содержит подслово $x = c$ для $c \in N \cup X$.

Выражения вида $WHILE (z \circledcirc \mu) DO (Q_2)$, фигурирующие в программах, назовем о п р е д е л е н н ы м и ц и к л а м и, а $(M)^\dagger$ - н е о п р е д е л е н н ы м и ц и к л а м и. В определенных циклах в условиях фигурируют интерпретированные предикаты $x \circledcirc \mu$; символ $\neq$ обведен с целью отделения его от других предикатных символов (например, $\neq$, $\le$, Emp), которые могут присутствовать в выражениях языка. Как и в языке многоточечных выражений ([2]) эти символы не имеют семантики (не интерпретированы) и учитываются синтезатором только в чисто синтаксическом плане. Отмеченная особенность языка на первый взгляд представляется излишней сложностью, од-

нако, она имеет определенное значение лишь для синтезатора, облегчая ему задачу синтеза, поскольку при превращении выражений языка в реальные программы все символы все равно необходимым образом интерпретируются.

В неопределенных циклах предикаты вообще не выделены - это связано с невозможностью их непосредственного синтеза по чисто синтаксической информации. По этим же причинам вместо обычных IF... THEN... ELSE (и CASE) в нашем языке фигурирует объединение $\cup$: предикаты здесь не отделены от действий; их отделение происходит только при интерпретации программы

Класс всех программ условимся обозначать через $\mathcal{P}$.

Проиллюстрируем теперь определение программы, построив программы для алгоритмов, примеры вычислений которых приведены в пункте I.

ПРОГРАММА I. Приведение матрицы к диагональному виду.

P_1: INPUT: array $a(i,j)$, $1 \le i \le m$, $1 \le j \le n$;

$i := 2$;

WHILE $(i \neq m)$ DO $((j := i-1)$ WHILE $(j \neq n)$ DO

$(a(i,j) := \dfrac{a(i,j)}{a(i,i-1)} \cdot a(i-1,i-1) - a(i-1,j)\ j^+)\ i^+)$

Если $i = m$, то STOP.

ПРОГРАММА 2. Сортировка посредством выделения максимального элемента.

P_2: INPUT: array $K(1), K(2), \ldots, K(m)$; $y := m$;

WHILE $(y \neq 1)$ DO $((x := 1)$ WHILE $(x \neq y)$ DO

$(K(x) \le K(x+1)$?; допустим нет;

$K(x) \longleftrightarrow K(x+1)\ x^+) \cup$

$(K(x)) \le K(x+1)$?; допустим да; $x^+)$

$M(y) \leftarrow K(y)\ ; y^-)$

Если $y = 1$, то $M(1) \leftarrow K(1)$; STOP.

ПРОГРАММА 3. "Быстрая" сортировка.

P_3 : INPUT : array $K(\underline{1}), K(\underline{2}), \ldots, K(\underline{m})$;

$x_0 := 1;\ y := m;$

стек: $= (x_0, y_0)$ WHILE (TEmp (СТЕК)) DO (x_0, y_0) := СТЕК);

WHILE $(x_0 \oplus y_0)$ DO (($x := x_0,\ y := y_0$)

WHILE $(x \oplus y)$ DO (

$(K(\underline{x}) \leqslant K(\underline{y})$?; допустим да; y^-)*

$K(\underline{x}) \leqslant K(\underline{y})$?; допустим нет; $K(\underline{x}) \leftrightarrow K(\underline{y})$; x^+

$(K(\underline{x}) \leqslant K(\underline{y})$?; допустим да; x^+)*

$K(\underline{x}) \leqslant K(\underline{y})$?; допустим нет; $K(\underline{x}) \leftrightarrow K(\underline{y})$; y^-)

стек: $= (\underline{x+1}, y_0)$; $y_0 := \underline{x-1}$;)).

Отметим, что подчеркнутые символы (например, WHILE, DO, $(\ ,\ )$) принадлежат алфавиту A и не имеют отношения к символам $(\ ,\ )$, WHILE, DO из OP. Однако, когда программа будет интерпретироваться, они, разумеется, приобретут одинаковый смысл.

3. МНОЖЕСТВО ЗНАЧЕНИЙ ПРОГРАММЫ

В рассматриваемых нами примерах (I–3) алгоритм разъясняется только на одном существенном фрагменте. В случае алгоритмов типа "разделяй и властвуй" их разъяснение вообще, как правило, ограничивается рассмотрением работы алгоритма только до момента первого разделения задачи на меньшие подзадачи. Учитывая это обстоятельство, мы намерены соответствующим образом определить значения программ – в определенном смысле это будут **неполные** примеры вычислений.

Пусть $R \in \mathcal{P}$. Применим к R следующую операцию **развертки**. Заменим внешний цикл WHILE(α) DO(T) на выражение TP, где P – специальный символ, не входящий в $A \cup X \cup OP$. Далее пусть C_2 определяется как в (2.I). Заменим

$$T = C_2 \text{WHILE}(\beta)\ \text{DO}(Q) \quad \text{на}$$

(3.I) $$C_2 \underbrace{QQ \ldots Q}_{n},$$

$n \in N$ и вставим C_2 перед P.

Далее заменим каждое выражение $(S)^*$ на конкатенацию

(3.2) $\underbrace{SS \ldots S}_{m}$,

где m – произвольное число из N, и затем заменим каждое вхождение $(\mathcal{U}_1 \cup \mathcal{U}_2 \cup \ldots \cup \mathcal{U}_k)$ на одно из слов $\mathcal{U}_i$. Наконец, между любыми двумя символами z^+ (или z^-) и $a \in \{w^+, w^- \mid w \in X\}$ вставим слово β. Полученное таким образом выражение $\bar{P}$ назовем (полной) разверткой P.

Например, для P_3 разверткой является

$\bar{P}_3$: INPUT: array $K(1), K(2), \ldots, K(m)$;

$x_0 := 1$; $y_0 := m$;

$STEK := (x_0, y_0)$ WHILE $(\neg Emp(STEK))$ DO (

$(x_0, y_0) := STEK$; $x := x_0$; $y := y_0$;

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим нет;

$K(x) \longleftrightarrow K(y)$; $x^+ \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $x^+ \; x \oplus y$

$K(x) \leq K(y)$?; допустим нет;

$K(x) \longleftrightarrow K(y)$; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $y^- \; x \oplus y$

$K(x) \leq K(y)$?; допустим нет;

$K(x) \longleftrightarrow K(y)$; $x^+ \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $x^+ \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $x^+ \; x \oplus y$

$K(x) \leq K(y)$?; допустим да; $x^+ \; x \oplus y$

$K(x) \leq K(y)$?; допустим нет;

$K(x) \leftrightarrow K(y)$ y^- $x \oplus y$

$(TEK := (\underline{x+1}, y_0); y_0 := \underline{x-1}; x := x_0, y := y_0; P)$.

Разверткой P будем называть также любое выражение, получаемое из полной развертки удалением фрагмента между некоторым $x \oplus y$ и последним словом $x \oplus y$ (включая последнее $x \oplus y$). Так, разверткой P_3 является

$\bar{P}_3'$: INPUT: array $K(1), K(2), \ldots, K(m)$;

$x_0 := 1$; $y_0 := m$;

$(TEK := (x_0, y_0)$ $\underline{WHILE}$ $(\neg Emp(TEK))$ $\underline{DO}($

$(x_0, y_0) := TEK$; $x := x_0$; $y := y_0$;

$K(x) \le K(y)$?; допустим да; y^- $x \oplus y$

$K(x) \le K(y)$?; допустим да; y^- $x \oplus y$

$K(x) \le K(y)$?; допустим да; y^- $x \oplus y$

$TEK := (\underline{x+1}, y_0)$; $y_0 := \underline{x-1}$; $x := x_0$; $y := y_0$; $P)$

Чтобы определить значение программы, перенумеруем все вхождения каждой переменной в $\bar{P}$, кроме СТЕК и вхождений типа $x :=$ и $y = \mu$ слева направо. Например, для P_3 получим

INPUT: array $K(1), K(2), \ldots, K(m^{(1)})$;

$x_0 := 1$; $y_0 := m^{(2)}$;

$TEK := (x_0^{(1)}, y_0^{(1)})$ $\underline{WHILE}$ $(\neg Emp(TEK))$ $\underline{DO}($

$(x_0, y_0) := TEK$; $x := x_0^{(2)}$, $y := y_0^{(2)}$;

$K(x^{(1)}) \le K(y^{(1)})$?; допустим да; $y^{(2)-}$ $x^{(2)} \oplus y^{(3)}$

$K(x^{(3)}) \le K(y^{(4)})$?; допустим да; $y^{(5)-}$ $x^{(4)} \oplus y^{(5)}$

$K(x^{(5)}) \le K(y^{(7)})$?; допустим да; $y^{(8)-}$ $x^{(6)} \oplus y^{(9)}$

$K(x^{(7)}) \le K(y^{(10)})$?; допустим да; $y^{(11)-}$ $x^{(8)} \oplus y^{(12)}$

$K(x^{(9)}) \le K(y^{(11)})$?; допустим нет;

$K(x^{(10)}) \le K(y^{(14)})$; $x^{(11)+}$ $x^{(14)} \oplus y^{(15)}$

и т.д.

Положим $v(c) = c$ для всех $c \in N$ и определим значение $v(t^{(i)})$ вхождения переменной $t \in X \setminus \{TEK\}$ с номером i:

1. $i = 1$.

Если слева от $t^{(i)}$ есть выражение $t:=\mu$, то $v(t^{(i)})=v(\mu)$, иначе полагаем $v(t^{(i)})$ равным некоторому $\alpha \in N$.

2. $i>1$.

$$v(t^{(i)}) \overset{\text{def}}{=} v(t^{(i-1)});$$

$$v(t^{(i)+}) \overset{\text{def}}{=} v(t^{(i-1)})+1;$$

$$v(t^{(i)-}) \overset{\text{def}}{=} v(t^{(i-1)})-1.$$

Выполним все действия вида $a-b$, $a+b$.

Так, для P_3 получим выражение:

INPUT: array K(1), K(2), . . , K(16);

$x_0:=1$; $y_0:=16$;

CTEK:=(1,16) WHILE (TEmp (CTEK)) DO (

$(x_0,y_0):=$CTEK; $x:=1$; $y:=16$;

K(1) ≤ K(16) ?; допустим да; 15 1⊕15

K(1) ≤ K(15) ?; допустим да; 14 1⊕14

K(1) ≤ K(14) ?; допустим да; 13 1⊕13

K(1) ≤ K(13) ?; допустим да; 12 1⊕12

K(1) ≤ K(12) ?; допустим нет;

K(1) ⟷ K(12); 2 2⊕12

K(1) ≤ K(12) ?; допустим да; 3 3⊕12

K(3) ≤ K(12) ?; допустим нет;

K(3) ⟷ K(12); 11 3⊕11

K(3) ≤ K(11) ?; допустим да; 10 3⊕10

K(3) ≤ K(10) ?; допустим да; 9 3⊕9

K(3) ≤ K(9) ?; допустим да; 8 3⊕8

K(3) ≤ K(8) ?; допустим нет;

K(3) ⟷ K(8); 4 4⊕8

K(4) ≤ K(8) ?; допустим да; 5 5⊕8

K(5) ≤ K(8) ?; допустим да; 6 6⊕8

K(6) ≤ K(8) ?; допустим да; 7 7⊕8

K(7) ≤ K(8) ?; допустим нет;

K(7) ⟷ K(8); 7 7⊕7

CTEK:=(8,16); $y_0:=6$; $x:=1$; $y:=6$; P)

(двухзначные числа подчеркнуты, чтобы отличить их от од-

нозначных).

Если в полученном выражении выполняются все неравенства $a \mathbin{\textcircled{\scriptsize$\neq$}} b$ (при стандартной интерпретации $\neq$ в $\textcircled{\scriptsize$\neq$}$) кроме последнего, то удалим все эти неравенства и значения выражений z^+, z^- для $z \in X$. Полученное выражение $v = v(P)$ назовем значением или примером вычисления программы P. Так, для P_3 из приведенного выше выражения получаем значение:

INPUT: array $K(1), \ldots, K(16)$;
$x_0 := 1$; $y_0 := 16$;
СТЕК := $(1, 16)$ WHILE (NEmp (СТЕК)) DO(
$(x_0, y_0) :=$ СТЕК; $x := 1$; $y := 16$,
$K(1) \le K(16)$?; допустим да;
$K(1) \le K(15)$?; допустим да;
$K(1) \le K(14)$?; допустим да;
$K(1) \le K(13)$?; допустим да;
$K(1) \le K(12)$?; допустим нет;
$K(1) \longleftrightarrow K(12)$;
$K(2) \le K(12)$?; допустим да;
$K(3) \le K(12)$?; допустим нет;
$K(3) \longleftrightarrow K(12)$;
$K(3) \le K(11)$?; допустим да;
$K(3) \le K(10)$?; допустим да;
$K(3) \le K(9)$?; допустим да;
$K(3) \le K(8)$?; допустим нет;
$K(3) \longleftrightarrow K(8)$;
$K(4) \le K(8)$?; допустим да;
$K(5) \le K(8)$?; допустим да;
$K(6) \le K(8)$?; допустим да;
$K(7) \le K(8)$?; допустим нет;
$K(7) \longleftrightarrow K(8)$;
СТЕК: $= (8, 16)$; $y_0 := 6$; $x := 1$; $y := 6$; P).

В свою очередь, примеры I и 2 являются примерами вычислений программ P_1 и P_2, а пример 3 является при-

мером вычисления для несколько иного варианта программы P_3.

К сожалению, в отличие от языка многоточечных выражений, синтез в рамках которого возможен по произвольным достаточно длинным примерам вычислений [2], в нашей модели примеры вычислений нуждаются в дополнительных аннотациях, поскольку в общей ситуации никакой синтезатор неспособен точно восстановить все переменные и связи между ними в программе. Аннотации в нашей модели имеют вид $[=x-c]$, $[=x+c]$, где $c \in N$ и x - переменная, а $[\ ] \notin A \cup OP$. Аннотациями снабжаются все значения переменных. Кроме того, если в программе в условии цикла фигурирует $x \neq y$, значения x во фрагменте, получаемом из последнего выражения Q в развертке (3.1), равны значениям y (или $y \pm 1$), то упомянутые значения x снабжаются аннотацией $[=y]$ (или, соответственно, $[=y \pm 1]$). В том случае, если Q не содержит неопределенных циклов и символов $\cup$, то допускается присутствие аннотаций не во всех Q в (3.1), а только в первом. Если же выражение в программе имеет вид (2.3), то аннотациями снабжаются все значения переменных в примере. Далее аннотациями в любом случае снабжаются все значения переменных z, не входящих в часть развертки $QQ \ldots Q$ (см. 3.1). Нетрудно видеть, что аннотации в примерах 1-3 соответствуют упомянутым требованиям. Аннотированный вариант примера вычисления программы P_3, приведенного выше, может быть следующим.

INPUT: массив $K(1), \ldots, K(16[=m])$;
$x_0 := 1$; $y_0 := 16[=m]$; $x := x_0$; $y := y_0$;
$CTEK := (1[=x_0]; 16[=y])$ WHILE $(\neg Emp(CTEK))$ DO (
$(x_0, y_0) := CTEK$; $x := 1[=x_0]$, $y = 16[=y]$;

$K(1[=x]) \leq K(16[=y])$?; допустим да;
$K(1[=x]) \leq K(15[=y])$?; допустим да;
$K(1[=x]) \leq K(14[=y])$?; допустим да;
$K(1[=x]) \leq K(13[=y])$?; допустим да;

$K(1[=x]) \leqslant K(\underline{12}[=y])$?; допустим нет;

$K(1[=x]) \longleftrightarrow K(\underline{12}[=y])$;

$K(2[=x]) \leqslant K(\underline{12}[=y])$?; допустим да;

$K(3[=x]) \leqslant K(\underline{12}[=y])$?; допустим нет;

$K(3[=x]) \longleftrightarrow K(\underline{12}[=y])$;

$K(3[=x]) \leqslant K(\underline{11}[=y])$?; допустим да;

$K(3[=x]) \leqslant K(\underline{10}[=y])$?; допустим да;

$K(3[=x]) \leqslant K(9[=y])$?; допустим да;

$K(3[=x]) \leqslant K(8[=y])$?; допустим нет;

$K(3[=x]) \longleftrightarrow K(8[=y])$;

$K(4[=x]) \leqslant K(8[=y])$?; допустим да;

$K(5[=x]) \leqslant K(8[=y])$?; допустим да;

$K(6[=x]) \leqslant K(8[=y])$?; допустим да;

$K(7[=x]) \leqslant K(8[=y=\underline{x+1}])$?; допустим нет;

$K(7[=x]) \longleftrightarrow K(8[=y=\underline{x+1}])$;

СТЕК := (8[= $\underline{x+1}$, 16[= y_0]); y_0 := 6[= $\underline{x-1}$]

x := 1[= x_0]; y := 6[= y_0]; P).

---

4. КАНОНИЧЕСКИЕ ПРОГРАММЫ И ПРЕДСТАВИТЕЛЬНЫЕ ПРИМЕРЫ

Для того, чтобы синтез программ по (неполным) примерам вычислений был успешным, программы, а также примеры вычислений должны удовлетворять некоторым естественным требованиям.

Сформулируем сначала требования, которым должны удовлетворять программы.

Нам понадобятся некоторые вспомогательные понятия.

Назовем слова $\alpha = a_1 a_2 \dots a_\kappa$ и $\beta = b_1 b_2 \dots b_m$ **существенно различными**, если для некоторого $i \leqslant \min(\kappa, m)$ $a_i \neq b_i$.

Пусть Q в программе P имеет вид (2.3) (см.определение программы):

$$R_0 (T_1 M_1)^* R_1 K_1 (T_2 M_2)^* R_2 K_2 \dots (T_m M_m)^* R_m K_m$$

Определим для Q три следующих преобразования:

Π_I. Пусть для некоторого слова α $R_i = R'_{i-1}\alpha$, $T_i = T'_i\alpha$. Тогда $\Pi_1(Q)$ получается заменой в Q слова

$$R_{i-1}\,K_{i-1}\,(T_i\,M_i)^*$$

на

$$R'_{i-1}\,K_{i-1}\,(\alpha\,T'_i\,M_i)^*\alpha$$

Π_2. Пусть для некоторого i $R_{i-1} = R'_{i-1}\,T_i$. Тогда $\Pi_2(Q)$ получается заменой R_{i-1} в Q на R'_{i-1}.

Π_3. Если Q представимо в виде $Q'Q'\ldots Q'$, то $\Pi_3(Q)$ получается заменой Q на Q'.

Если к программе P (точнее, к выражению Q в ней) ни одно преобразование Π_I, Π_2, Π_3 не применимо, назовем P к а н о н и ч е с к и м. Нетрудно видеть, что практически все "разумные" программы можно считать каноническими.

В дальнейшем мы будем рассматривать только канонические программы, удовлетворяющие следующему условию:

(а) все слова T_i и R_j, $1 \le i, j \le m$, в Q попарно существенно различны.

Условие (а) достаточно естественно, поскольку различные T_i и (R_i) соответствуют различным значениям предиката в условиях циклов, определяющих Q (см. приведенные выше примеры программ P_I-P_3).

Класс всех канонических программ удовлетворяющих условию (а), обозначим через $\mathcal{P}_0$.

Определим теперь класс примеров, по которым удобно синтезировать программы из $\mathcal{P}_0$.

Назовем развертку $\bar{P}$ программы P п р е д с т а в и т е л ь н о й, если при построении $\bar{P}$ $n \ge 2$ в (3.I) и если Q имеет вид (2.2), то для каждого i, $1 \le i \le m$ в развертке $\bar{P}$ фигурирует подслово $T_i M_i \bar{T}_i M_i$; если же Q имеет вид (2.2), то каждое $(T_i M_i)^*$ заменяется в $\bar{P}$ на

$$\underbrace{T_i M_i \ldots T_i M_i}_{q}$$

с $q \geq 2$.

Назовем аннотированный пример вычисления программы п р е д с т а в и т е л ь н ы м, если он соответствует представительной развертке.

В следующем разделе представлен алгоритм синтеза, который по произвольному представительному аннотированному примеру канонической программы P синтезирует программу, эквивалентную P.

5. АЛГОРИТМ СИНТЕЗА

Метод синтеза программ из $\mathcal{P}_0$ достаточно естественен: во входном слове ищется самый левый фрагмент, который может быть значением цикла; найденный фрагмент заменяется циклом, и процедура повторяется.

Опишем алгоритм синтеза более подробно. Пусть w - входной пример. Действия алгоритма разделяются на три этапа: на первом этапе синтезируется выражение Q (см. (2.I)), на втором - цикл

(5.I) $C_2\, WHILE\ (y \oplus v)\ DO\ (Q)$

и на третьем - выражение (2.I).

Рассмотрим сначала первый этап. Здесь возможен случай, когда не все значения переменных снабжены аннотациями (см. пример I). В этом случае нужно в w найти подслово $w_1 w_2$ такое, что после удаления аннотаций из $w_1 w_2$

$$w_1 = a_1 a_2 \ldots a_k$$
$$w_2 = b_1 b_2 \ldots b_k ,$$

и если символам a_i, a_j была в w_1 приписана аннотация (скажем, $[=x]$), то либо $b_i = a_i = b_j = a_j$, либо

$$\alpha_i - a_i = \alpha_j - a_j = \mp 1.$$

В качестве T в $Q = TM$ тогда следует взять w_1, заменив в нем символы a_i, имевшие аннотации $[=x+c]$ на $[x+c]$ (для $[=x]$ – просто на x), а в качестве M – слово, содержащее x^+, если значения x возрастают в w_2, и x^-, если значения x в w_2 меньше, чем в w_1.

Рассмотрим теперь ситуацию, когда все значения переменных в w имеют аннотации. Здесь процедуру синтеза можно разделить на конечное число шагов. Пусть к шагу ρ уже определены некоторые выражения $T_1, T_2, \dots, T_i$ и $R_0, R_1, \dots, R_j$, и под некоторым символом a_c в w установлена метка $\square$.

Шаг ρ. Если метка установлена справа от w_1, то прекращаем процедуру. Иначе ищем такое подслово $a_0 a_1 \dots a_\kappa$, которое при замене значений переменных на соответствующие аннотации, совпадает с каким-нибудь T_s. Если такое $a_1 \dots a_\kappa$ найдено, то передвигаем метку на $a_{\kappa+1}$ и переходим к шагу $\rho+1$. Если такое подслово не найдено, ищем самое левое подслово $\bar{a} = a_1 a_2 \dots a_\kappa$ справа от метки $\square$ наименьшей длины, удовлетворяющее следующему условию:

(*) в w существует такое подслово $w_1 w_2$, что

(а) при замене в любом w_i и в $\bar{a}$ значений переменных на соответствующие аннотации полученные слова $\bar{a}'$ и w_i' совпадают;

(б) $w_1 \neq w_2$.

Если требуемое $\bar{a}$ не найдено, то заменяем в части слова w справа от $\square$ значения переменных на соответствующие аннотации и процедуру завершаем. Иначе заменяем значения переменных на аннотации во фрагменте w между $\square$ и $\bar{a}$ и обозначаем его через R_{j+1}, обозначаем $\bar{a}'$ через T_{i+1}, передвигаем $\square$ на a_{i+1} и переходим к шагу $\rho+1$.

Итак, все T_i и R_i в Q определены. M_i и K_i (см.(2.2)) определяются очевидным образом: если x воз-

растает, то в M_i заносится x^+, если убывает - x^-. То же - для K_i.

Теперь нужно определить, является ли Q объединением или имеет вид (2.3). Если хотя бы одно $R_i \neq \Lambda$, то Q имеет вид (2.3) (который получается достаточно очевидно), иначе Q является объединением.

На втором этапе формируется цикл (5.I). Переменные для условия $y \oplus \nu$ берутся из аннотаций вида $[=x=y=c]$ (см. Примеры 2 и 3).

На третьем этапе формируется выражение (2.I). Для определения условия $x \oplus \mu$ находим переменную x, фигурирующую в аннотациях во фрагменте w, отвечающем Q, которой перед этим фрагментом и после него присваиваются значения, отличающиеся на I (переменная i в примере I, переменная y в примере 2), либо переменную, не фигурирующую в аннотациях во фрагменте, соответствующем Q, которой перед и после этого фрагмента присваиваются различные значения (y_0 в примере 3). В качестве μ в первом случае выбираем либо переменную, фигурирующую в аннотации для выражения $x=c$ после символа P (такое выражение фигурирует в примере вычисления в силу определения программы), либо сам символ c, если аннотация отсутствует (см. примеры I и 2). Во втором случае полагаем μ равным c в $x=c$ после P.

Теперь нужно определить присваивания, входящие в C_2. Если в Q фигурирует z^+ или z^-, то перед фрагментом, соответствующим Q, в w должен находиться оператор $z:=c$. Если c приписана аннотация $[=\delta]$, то в C_2 заносится оператор $z:=\delta$; иначе $z:=c$.

Теперь остается заменить оставшиеся в выражении числа, снабженные аннотациями, на эти аннотации, и на этом синтез завершается.

Нетрудно видеть, что по примерам I,2,3 наш алгоритм синтезирует программы P_1, P_2 и, соответственно, P_3.

Корректность алгоритма синтеза вытекает из следующего утверждения. Пусть $\mathcal{A}$ - наш алгоритм синтеза.

ТЕОРЕМА. По любому представительному аннотированному примеру вычисления w программы $P \in \mathcal{P}_0$ алгоритм $\mathcal{A}$ находит программу $P(w)$, эквивалентную P.

ДОКАЗАТЕЛЬСТВО. Мы приведем лишь существенные моменты доказательства, опуская технические детали.

Рассмотрим произвольный шаг p на первом этапе построения $P(w)$. Выбор самого левого подслова $\bar{a}$ с требуемыми свойствами обусловлен неприменимостью к P операций Π_1 и Π_2. Выполнение условия (а) обуславливает однозначность выбора "шаблонов" T_i и R_j; длина T_i однозначно обуславливается представительностью примера w. Выбор "самого короткого" Q вызван неприменимостью к P операции Π_3. Определения на этапах 2 и 3 достаточно очевидны.

Заметим, что примеры, по которым осуществляется синтез, имеют длину, сравнимую с длиной самой программы P. Сам алгоритм синтеза имеет, очевидно, полиномиальную сложность (от длины входного примера). Это дает основания надеяться на построение достаточно эффективного практического синтезатора (при добавлении соответствующей эвристики).

Автор благодарит проф. Я.М.Барздиня за ценное обсуждение.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Барздинь Я.М. Один подход к проблеме индуктивного вывода // Применение методов математической логики: Тезисы докл. III Всесоюзной конф. Таллин, 1983. С.16-28.
2. Barzdin J.M. Some rules of inductive inference and their use for program synthesis // Inf. Processing 83. North-Holland, 1983. p.333-337.
3. Бразма А.Н., Кинбер Е.Б. Язык для синтеза программ с ветвлениями нелинейного типа, содержащих циклы "do while" // Семиотические аспекты формализации интеллектуальной деятельности: Тезисы докл. Дубна, ВНИИТИ. 1985. С.173-176.

4. Brazma A.N., Kinber E.B. Generalized regular expressions - a language for synthesis of programs with branching in loops // Theor.Comp.Sci. V.46. 1986. P.175-195.

5. Кнут Д. Искусство программирования на ЭВМ. М., Мир, 1978. Т3.

БУЛЕВЫ ФУНКЦИИ НА БЕСКОНЕЧНЫХ СЛОВАХ

М.Я.Алберто

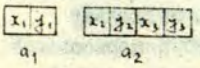
ВЦ при ЛГУ им.П.Стучки

Рассмотрим вычисление булевых функций с бесконечно многими переменными $x_1, x_2, \ldots, x_n, \ldots$ или распознавание бесконечных слов (w-слов) вида $x_1, x_2, \ldots, x_n, \ldots$, где $x_i \in \{0,1\}$. Множеством предельных состояний будем называть множество тех состояний $q_i, \ldots, q_i$, которые при данной реализации машины Тьюринга или конечного автомата повторяются бесконечно много раз. Множество предельных состояний и определяет, принадлежит ли w-слово w-языку или нет. Рассмотрим некоторый ω-язык L. Будем говорить, что буква x_i нефиктивная, если существуют такие $z_1, \ldots, z_{i-1}, z_{i+1}, \ldots$, что слова $z_1 z_2, \ldots, z_{i-1} 0 z_{i+1} \ldots$ и $z_1 z_2 \ldots z_{i-1} 1 z_{i+1} \ldots$ разные по отношению принадлежности к w-языку L.

Если односторонняя машина M на w-слове $x_1 \ldots x_i \ldots$ работает по-разному после прочтения буквы x_i в зависимости от значения x_i, то будем говорить, что M читает букву x_i. Определим следующую меру сложности. Будем говорить, что сложность вычислений машины M есть r(n), если при любом $x_1, x_2, \ldots, x_n, \ldots$ среди первых n букв не больше r(n) букв прочтено.

Определим w-язык A следующим образом.

В слове вида



определим некоторую процедуру хождения по сегментам $a_1, a_2, a_3, \ldots$

В сегменте a_2 рассмотрим x_{i_2} (x_2, если $x_1 = 0$; x_3, если $x_1 = 1$), в сегменте a_3 рассмотрим x_{i_3} (x_4, если $x_1x_{i_2} = 00$; x_5, если $x_1x_{i_2} = 01$; x_6, если $x_1x_{i_2} = 10$; x_7 , если $x_1x_{i_2} = 11$) и т.д. w-слово принадлежит w-языку, если имеет место $y_1\, y_{i_2}\, y_{i_3}\, y_{i_4} \ldots = 1111 \ldots$

ТЕОРЕМА I. I) Для w-языка A любая буква нефиктивна;

2) односторонняя машина Тьюринга распознает A со сложностью $\log n$.

ДОКАЗАТЕЛЬСТВО. I) Рассмотрим отдельно четные и нечетные места букв. Без пояснений даем два примера

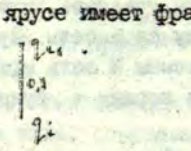
01 1100 000^0_1 1111111111 .....

01 $0^0_1$100 0001 1111111111 .....

2) Машина M, распознающая A, работает следующим образом. При чтении y_1 M в состоянии q_2. Если M встречает первого $y_{i_j} = 0$, то она попадает в состояние q_3 и в q_2 никогда не возвращается. Акцептирующими являются все подмножества состояний, которые содержат q_2.

ТЕОРЕМА 2. Если для некоторого языка Z все буквы нефиктивны, то нельзя его распознавать односторонней машиной Тьюринга со сложностью меньшей, чем $\log n$.

ДОКАЗАТЕЛЬСТВО. Рассмотрим вычисления машины Тьюринга M на всевозможных w-словах в алфавите $\{0,1\}$, которые отображены в виде бесконечного бинарного дерева $\mathcal{U}$. Если M в состоянии q_{i_1} читает букву x_j из w-слова $x = x_1x_2\ldots$, то дерево $\mathcal{U}$ на j-ом ярусе имеет фрагмент



в противном случае –

Зафиксируем некоторое n. Рассмотрим дерево $\mathcal{U}$ до (n+1)-го яруса включительно. Получим некоторое конечное дерево $\mathcal{U}'$, которое содержит не меньше n ветвлений, поскольку все буквы нефиктивны. Нетрудно убедиться, что бинарное дерево, которое имеет n ветвлений, содержит ветвь длиной

$\log n$. Это означает, что при некотором w-слове x_0 машина M читает не меньше $\log n$ букв из первых n.

Рассмотрим некоторую последовательность w-языков D, D_1, D_2,... , для которых нефиктивных букв нет, и которые можно распознавать со сколь угодно малой возрастающей сложностью.

$D = \{ x_1 \ldots \mid$ для бесконечно многих i $x_i = 1 \}$, $r(n) = n$;

$D_1 = \{ x_1 \ldots \mid$ для бесконечно многих $i \in \{2, 2^2, 2^3, \ldots\}$ $x_i = 1 \}$, $r(n) = \log n$;

$D_2 = \{ x_1 \ldots \mid$ для бесконечно многих $i \in \{2^2, 2^{2^2}, 2^{2^3}, \ldots\}$ $x_i = 1 \}$, $r(n) = \log \log n$;

.....

Здесь каждая буква сама по себе ничего не решает, но некоторое подмножество существенно. Какой ее объем?

Множество I будем называть нефиктивным для w-языка L, если существует такая последовательность значений $\alpha^0 = \{\alpha_i\}$ для всех $i \in \overline{I}$ и последовательности $\alpha^1 = \{\alpha_i\}$ для $i \in I$ и $\alpha^2 \in \{\alpha_i\}$ для $i \in I$, что $\alpha^0 \cup \alpha^1$ принадлежит w-языку L, а $\alpha^0 \cup \alpha^2$ не принадлежит w-языку L.

Множество I будем называть фиктивным для w-языка L, если для каждой последовательности $\alpha^0 = \{\alpha_i\}$ для всех $i \in \overline{I}$ все слова вида $\alpha^0 \cup \alpha$, где $\alpha = \{\alpha_j\}$, $j \in I$, либо принадлежат w-языку L, либо не принадлежат w-языку L. Без доказательства приведем леммы 3, 4, которые характеризуют свойства фиктивных множеств.

ЛЕММА 3. 1) Если множество I фиктивно для L, то каждое множество J, где $J \subset I$, фиктивно.

2) Если I и J фиктивные множества для L, то $I \cup J$ фиктивное множество для L.

ЛЕММА 4. 1) Если w-язык L нетривиальный и множество I фиктивное, то $\overline{I}$ нефиктивное.

2) Существует w-язык L и нефиктивные множества I и J для J, для которых $I \cap J$ содержит бесконечно много букв и является фиктивным.

Сейчас введем меру: среди первых n букв имеется не больше f(n) нефиктивных букв.

Множество I будем называть f-жидким, если среди первых n букв не более f(n) являются нефиктивными. Это означает: среди первых n букв выделено такое фиктивное множество J для L, что среди первых n букв в J находятся не более, чем $f(n)$ элементов.

ЛЕММА 5. Для любой возрастающей функции f(n) неправда, что D является (n-f(n))-жидким множеством.

ДОКАЗАТЕЛЬСТВО. От противного: существует такая возрастающая f(n), что выделено нефиктивное бесконечное множество J. Тогда заполняем J нулями и, в зависимости от того, сколько "1" во множестве J, это w-слово принадлежит или не принадлежит w-языку D. Это означает, что при распознавании w-языка D односторонней машиной надо задать почти все вопросы.

ТЕОРЕМА 6. Для любого $c > 0$ существует односторонний алгоритм, который распознает w-язык D с вероятностью 1 и задает не более, чем $c \cdot n$ вопросов.

ДОКАЗАТЕЛЬСТВО. Предположим, что мы выбрали большую константу k в зависимости от c. Наш односторонний вероятностный алгоритм работает следующим образом. Перед каждым вопросом бросаем монету, которая с вероятностью $\frac{1}{k}$ выдает ответ "да" и с вероятностью $1 - \frac{1}{k}$ выдает ответ "нет". В случае ответа "да" спрашиваем букву x_i, если "нет", то не спрашиваем. Среди всех состояний q_2 является особым: в это состояние попадает машина, если прочитанная буква $x_j=1$. Подмножество состояний является акцептирующим т.и.т.т., если в нее входит состояние q_2.

Возможны два случая:

1) w-слово содержит конечное число "1" - в этом случае машина выдает правильный ответ;

2) w-слово содержит бесконечно много "1". Будут ли в этом случае с вероятностью 1 опрошено бесконечно много букв, содержащих "1"?

Обозначим номера букв, равных "1", через $i_1, i_2, i_3, \ldots$ При $i=i_1, i_2, \ldots$ бросаем монету. В терминах леммы Бореля-Кантелли $p_1 = \frac{1}{k}$, $p_2 = \frac{1}{k}$, $p_3 = \frac{1}{k}, \ldots$

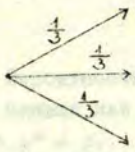
Ряд $\sum_{i=1}^{\infty} p_i$ расходится. Это означает, что из леммы Бореля-Кантелли с вероятностью "1" выдается ответ "да" бесконечно много раз.

Оценку числа вопросов $c \cdot n$ с вероятностью "1" можно получить из закона повторного логарифма, который гласит, что при $\lambda > 1$ с вероятностью 1 происходит только конечное число событий $S_n > n \cdot p + \lambda\sqrt{2\,npq\,\log\log n}$ [1]. Точное доказательство этого факта здесь приводить не будем.

Рассмотрим язык $C = \{ x_1 \ldots \mid$ для любого i $x_i = 1 \}$. Очевидно, что при детерминированном алгоритме необходимо задавать все n вопросы.

Сейчас рассмотрим несколько вероятностных алгоритмов, в которых монета бросается только однажды в начале работы, для распознавания языков C(D).

01 1 C(D)



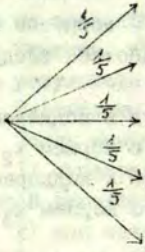
проверка условия "$x_i = 1$" для четных i.

проверка условия "$x_i = 1$" для нечетных i.

нет (да)

Правильный ответ получаем с вероятностью $\frac{2}{3}$. Число вопросов $\frac{n}{2}$.

2 C(D)



проверка условия $x_i=1$ для i, где i mod4 = 0,1,2

проверка условия $x_i=1$ для i, где i mod4 = 0,2,3

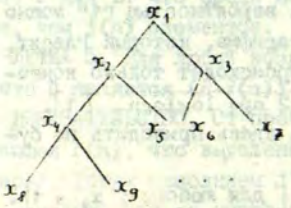
проверка условия $x_i=1$ для i, где i mod4 = 0,1,3

проверка условия $x_i=1$ для i, где i mod4 = 1,2,3

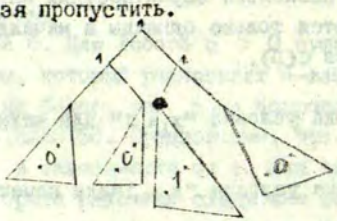
нет (да)

Правильный ответ получаем с вероятностью $\frac{4}{5}$. Число вопросов $\frac{3}{4}$ n.

Определим w-язык E. w-слово $x_1, x_2, \ldots$ принадлежит w-языку E т.и.т.т., если существует путь, содержащий только "1" в бинарном дереве



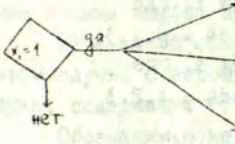
Детерминированный алгоритм требует n вопросов - ни одного нельзя пропустить.



В случае пропуска x_6 можем получить неправильный результат.

Недетерминированный алгоритм угадывает этот путь, который требует $\log n$ вопросов.

Вероятностный алгоритм, который задает $\frac{n}{2}$ вопросов и выдает правильный ответ с вероятностью $\frac{2}{3}$, работает следующим образом:

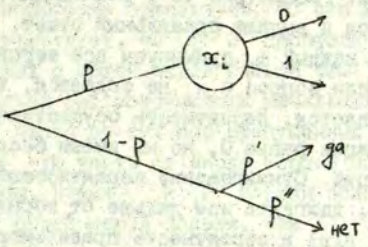


проверка наличия "1"-пути в поддереве с корнем x_2.

проверка наличия "1"-пути в поддереве с корнем x_3.

ТЕОРЕМА 7. Не существует функция $g(n) = o(n)$ такая, что язык E распознается некоторым вероятностным алгоритмом, который на первых n буквах задает не более, чем $g(n)$ вопросов.

ДОКАЗАТЕЛЬСТВО. Рассмотрим работу алгоритма на таком дереве, где всэ буквы до k-го яруса равны "1", все сыновья некоторой вершины x_i k-го яруса равны "1", все сыновья всех остальных вершин k-го яруса равны "0". Тогда можем утверждать следующее. Если сумма вероятностей по всем реализациям спросить значение x_i меньше, чем p, то вероятность выдачи правильного результата не больше $\frac{1+p}{2}$.



Вероятность выдачи правильного результата в случае "да" меньше или равна p' + p , в случае "нет" меньше или равна p" + p.

$$\min\{p', p''\} \leqslant \frac{1-p}{2}$$

Следовательно, вероятность выдачи правильного результата не превосходит $\frac{1-p}{2} + p = \frac{1}{2} + \frac{p}{2}$.

Отметим, что в случае $p=\frac{1}{2}$ вероятность выдачи правильного результата не больше $\frac{3}{4}$.

Рассмотрим работу алгоритма на чистые "1".

Тогда имеет место одно из двух:

I) существует вершина x_i некоторого k-го яруса, для которой вероятность задать вопрос меньше $p=\frac{1}{2}$;

2) для всех x_i вероятность задать вопрос больше p.

В случае I) вероятность выдачи правильного результата меньше $\frac{3}{4}$.

В случае 2) оценим сумму вероятностей p'спросить вершины k-го яруса и сумму вероятностей p" спросить все вер-

шины до(k-1)-го яруса.

$$2^k \cdot \frac{1}{2} < p' < 2^k$$

$$(2^k - 1) \cdot \frac{1}{2} < p'' < 2^k - 1 .$$

По предположению существует алгоритм, который задает $\sigma(n)$вопросов и выдает правильный ответ с вероятностью $\frac{3}{4}$.

Сейчас подсчитаем вероятности только по тем $\frac{3}{4}$ "хорошим" реализаций. Рассмотрим те $\frac{3}{4}$ реализаций, которые задают $\sigma(n)$ вопросов и выдают правильный ответ. Про каждую реализацию и про каждый x_i суммируем все вероятности задать вопрос о x_i. Если вопрос о x_i не задается, то эта вероятность не учитывается. Вероятность осуществления одной конкретной реализации равна 0, но мы имеем бесконечно много разных реализаций. Суммирование вероятностей производим по ярусам до k-го, двигаясь все дальше от корня дерева. Если число вопросов $\sigma(n)$ и вероятность правильного ответа $\frac{3}{4}$, то для больших k нас интересующая сумма $\sum$ меньше, чем $\frac{3}{4} \cdot \sigma(2^k)$.

На "плохих" реализациях вероятность задать вопросы про 2^k буквы меньше, чем $\frac{1}{4} \cdot 2^k$, где 2^k - число всевозможных вопросов.

Следовательно, $(2^k - 1) \cdot \frac{1}{2} < \frac{3}{4} \cdot \sigma(2^k) + \frac{1}{4} \cdot 2^k < 2^k - 1$. Противоречие.

Далее рассмотрим язык F, который от языка E отличается тем, что требуется существование пути, содержащего бесконечное число "1".

Легко видеть, что язык F распознается детерминированным образом, не задавая бесконечно много вопросов, например, по одному на каждом пути. Если дерево содержит только "0", то ясно, что ни на одном пути нельзя пропустить бесконечно многих вопросов. Найдем k-тый ярус, ниже которого все вопросы задаются. Это означает, что $\frac{1}{2^k} \cdot n$ вопросов необходимо.

ТЕОРЕМА 8. Не существует такой детерминированный алгоритм и такой $\varepsilon > 0$, что, начиная с некоторого n_0, для

всех $n > n_0$ не задается больше, чем $(1-\varepsilon)n$ вопросов для распознавания языка F.

ДОКАЗАТЕЛЬСТВО. Идея доказательства состоит в следующем. Предположим от противного, что такой алгоритм существует. Рассмотрим произвольное натуральное число $n_1 > n_0$ и произвольное натуральное число l. Покажем, что существует такой путь в дереве, что на этом пути не заданы l вопросов. Рассмотрим столь большой начальный фрагмент дерева, чтобы число вершин в нем было больше n_1. Если путь с больше чем l не заданными вопросами существует, то все хорошо. Если такой путь не существует, то $\varepsilon \cdot 2^k$ пропущенные вершины содержатся в путях с общей мерой больше чем ε.

Потом увеличим рассматриваемый начальный фрагмент дерева так, чтобы число до сих пор пропущенных вершин не превосходило $\frac{\varepsilon}{2}$ -ую часть от числа вершин нового фрагмента. Таким образом, новых $\frac{\varepsilon}{2} \cdot 2^k$ вершин прибавилось и они входят в пути с общей мерой больше, чем $\frac{\varepsilon}{2}$. Таким образом, больше, чем в $\frac{1 \cdot 2}{\varepsilon}$ раз увеличивали длину фрагмента.

Покажем, что в среднем на одном пути больше чем l раз встречаются такие вершины. Мера пропущенных вершин больше, чем $\frac{1 \cdot 2}{\varepsilon} \cdot \frac{\varepsilon}{2}$. Это означает, что на пути, где плотность пропущенных вершин максимальная, она не меньше l.

Таким образом, существует конечный путь с не меньше, чем одной пропущенной вершиной, существует конечный путь с не меньше, чем двумя пропущенными вершинами и т.д.

Следовательно, можем построить путь с бесконечно многими пропущенными вершинами. Противоречие.

Рассмотрим последовательность положительных чисел $\{p_i\} = \left\{\frac{1}{i \cdot \log i}\right\}$. Известно, что ряд $\sum_{i=1}^{\infty} \frac{1}{i \cdot \log i}$ расходится. Рассмотрим всевозможные множества натуральных чисел I. В зависимости от I ряд $\sum_{i \in I} \frac{1}{i \cdot \log i}$ может как сходиться, так и расходиться.

Определим язык G следующим образом. G содержит те и только те w-слова $\alpha_1 \alpha_2 \alpha_3 \cdots$ из $\{0,1\}^*$, для которых расходится ряд $\sum_{i \in I} p_i$, где I множество тех i, для которых $\alpha_i = 1$.

ТЕОРЕМА 9. Существует вероятностный алгоритм, который распознает язык G с вероятностью 1 и задает $\sigma(\log n)$ вопросов.

ДОКАЗАТЕЛЬСТВО. Вероятностный алгоритм работает следующим образом. Для каждого $i=1,2,3,\ldots$ с вероятностью p выдает ответ "да", с $1-p$ "нет". Если ответ "нет", то вопрос о значении x_i не задается и мы переходим к следующему x_{i+1}. Если ответ "да", то алгоритм переходит в состояние q_1, если $x_i=1$ и в состояние q_2, если $x_i=0$. Далее алгоритм действует одинаково в обоих случаях.

Все подмножества состояний, для которых q_1 предельное состояние, являются акцептирующими.

Будет ли задано бесконечно много вопросов в тех местах, где $x_i=1$?

Следуя лемме Бореля-Кантелли: либо с вероятностью 1 задается бесконечно много вопросов, либо с вероятностью 1 задается только конечное число вопросов в зависимости от того, расходится или сходится ряд $\sum_{i\in I} p_i$. Так как для конкретного w-слова соответствующее множество I обладает одним из вышеуказанных свойств, то с вероятностью "1" w-слова акцептируется или отвергается. Точную оценку числа вопросов здесь проводить не будем. Заметим только, что эта оценка производится техникой, похожей на ту, которая используется при доказательстве закона двойного логарифма [I].

Нетрудно убедиться в справедливости следующей теоремы.

ТЕОРЕМА IO. Не существует такой детерминированный алгоритм и такой $\varepsilon > 0$, что, начиная с некоторого n_0, для всех $n > n_0$ не задается больше, чем $(1-\varepsilon)n$ вопросов для распознавания языка G.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

I. В.Феллер. Введение в теорию вероятностей и ее приложения. М.: Мир , 1984.

2. D.Perrin. Recent results on automata and infinite words // Lecture notes in Computer Science, Springer-Verlag, 1984. V.176. P.134-148.

3. Ben Or.M. Lower bounds for algebraic computation trees // Proc. 15th ACM STOC Symposium, 1983. P.80-86.

ВЕРОЯТНОСТНЫЕ АЛГОРИТМЫ И РАСПОЗНАВАЕМОСТЬ ПРОДОЛЖАЕМОСТИ СЛОВ ДО ЯЗЫКА

Р.В.Фрейвалд
ВЦ при ЛГУ им. П.Стучки

М.П.Хитил
Карлов университет Прага ЧССР

В настоящей работе продолжается накопление материала для сравнения сложности двух близких, но существенно различных задач. Пусть дан некоторый язык L в конечном алфавите. Одна задача - это распознавание языка L, т.е. поиск ответов на вопросы "принадлежит ли данное слово x языку L ?" Другая задача - это продолжаемость слов до языка L, т.е. поиск ответов на вопросы "может ли данное слово x быть продолжено до некоторого слова $y \in L$?"

В [1] доказано, что, если для некоторого языка L первая задача разрешима детерминированной односторонней магазинной машиной, то и вторая задача разрешима детерминированной односторонней магазинной машиной. В [2] это утверждение усиле-

но там показано, что машину для решения второй задачи всегда можно представить как подмашину первой машины.

Ниже изучается вопрос, сохраняется ли подобное утверждение, если вместо детерминированных машин рассматриваются вероятностные. Получен ответ: аналог этого утверждения не имеет места. Однако при этом выяснена любопытная зависимость от вероятности выдачи машинами правильного результата.

Далее рассмотрим только частный случай магазинных машин - счетчиковые машины.

Построен такой язык L_2, который сам можно распознавать на вероятностных односторонних счетчиковых машинах со сколь угодно высокой вероятностью $1-\varepsilon$ ($\varepsilon > 0$), но для которого продолжаемость слов до L_2 на вероятностных односторонних счетчиковых машинах можно распознавать с вероятностью 2/3 и нельзя распознавать с вероятностью, превышающей 2/3. Построен такой язык L_3, который сам можно распознавать на таких машинах со сколь угодно высокой вероятностью $1-\varepsilon$ ($\varepsilon > 0$), но для которого продолжаемость слов до L_2 на упомянутых машинах можно распознавать с вероятностью 3/5 и нельзя распознавать с вероятностью, превышающей 3/5. Языки L_2 и L_3 являются элементами построенной в настоящей работе последовательности языков $\{L_k\}_{k=1,2,3,\dots}$, в которой каждый язык L_k можно распознавать с вероятностью $1-\varepsilon$ ($\varepsilon > 0$), но продолжаемость слов до языка L_k можно распознавать с вероятностью к/2к-I и нельзя распознавать с вероятностью, превышающей к/2к-I. Наконец, построен язык L_0, который можно распознавать со сколь угодно высокой вероятностью $1-\varepsilon$ ($\varepsilon > 0$), но для которого продолжаемость слов до L_0 алгоритмически неразрешима.

Техника доказательства, используемая в работе, существенно использует методы, разработанные в /3/.

Приведем необходимые определения.

Детерминированная односторонняя счетчиковая машина - это шестерка $\langle X, S, q_1, F_0, F_1, I \rangle$, где

X - конечный алфавит (букв на ленте), включающий символ $\#$;

S - конечный алфавит (внутренних состояний);

q_1 - инициальное состояние ($q_1 \in S$);

F_0 - множество принимающих состояний ($F_0 \subseteq S$);

F_1 - множество отвергающих состояний ($F_1 \subseteq S$);

I - множество инструкций. Каждая инструкция является цепочкой символов из $X \times S \times \{0,1\} \times \{\rightarrow\} \times S \times \{C,R\} \times \{-1,0,+1\}$, причем, если первый символ инструкции является #, то 6-й символ инструкции равен C. Первые 3 символа называются ее левой частью, а последние 3 - ее правой частью. Требуется, чтобы для любой цепочки из $X \times (S \setminus (F_0 \cup F_1)) \times \{0,1\}$ во множестве I нашлась одна и только одна инструкция с такой левой частью.

У автомата имеется одна лента, на которой написано рассматриваемое слово. Непосредственно после последней буквы этого слова написана одна буква #. Еще у автомата имеются n счетчиков, в каждый из которых может быть записано произвольное целое число. Автомат начинает работу в состоянии q_1. Головка расположена на первой с левой стороны букве распознаваемого слова. Счетчик содержит число 0. Пусть на очередном шаге автомата состояние равно $q_k \in S$, головка обозревает букву $x \in X$, и счетчик содержит число k. Через ξ обозначим число 0, если $k = 0$, и число 1, если $k \neq 0$. Пусть множество I содержит инструкцию $x q_k \xi \rightarrow q_{k'} \zeta \eta$, где $\zeta \in \{C,R\}$ и $\eta \in \{-1,0,+1\}$. Тогда автомат на этом шаге переходит в состояние $q_{k'}$; если $\zeta = C$, то головка остается на ленте, и если $\zeta = R$, то головка двигается по ленте на одну ячейку вправо; к содержанию счетчика прибавляется число η. Автомат кончает работу, когда вырабатывающееся новое внутреннее состояние принадлежит $F_0 \cup F_1$. Данное слово принимается, если это состояние принадлежит F_0 и отвергается, если - принадлежит F_1.

Определение вероятностной односторонней счетчиковой машины отличается от определения детерминированной односторонней счетчиковой машины лишь тем, что в левой части инструкции появляется еще один символ - выходное значение

датчика случайных чисел с конечным алфавитом, который на каждом шаге работы выдает свои выходные значения равновероятно и по схеме Бернулли, т.е. независимо от значений, выданных на других шагах. Требование, чтобы для каждой возможной левой части инструкции (дополненной теперь еще одним символом) во множестве I нашлась одна и только одна инструкция с такой левой частью, сохраняется.

Для каждой заканчивающейся реализации работы вероятностной машины можно подсчитать вероятность этой реализации. Вероятность результата y при работе машины на x – это сумма вероятностей реализаций работы машины на x, при которых вырабатывается результат y.

Говорят, что вероятностная машина распознает язык L с вероятностью p ($p > 1/2$), если машина при работе на произвольном x с вероятностью, не меньшей чем p, принимает x, если $x \in L$, и отвергает x, если $x \in \bar{L}$.

Языки $L_1, L_2, \ldots$ определяются так. Язык L_k состоит из слов в алфавите $\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_n\}$. Все слова языка L_k имеют вид $a_1^{n_1} a_2^{n_2} \ldots a_n^{n_k} b_1^{n_1} b_2^{n_2} \ldots b_n^{n_k}$, где $n_1, n_2, \ldots, n_k \in N^+$.

ПРЕДЛОЖЕНИЕ I. Для любого $k \in N^+$ и для любого $\varepsilon > 0$ язык L_k можно распознавать на вероятностной односторонней счетчиковой машине с вероятностью $1-\varepsilon$ (притом так, что на словах из L_k правильный ответ выдается с вероятностью 1).

ДОКАЗАТЕЛЬСТВО опирается на свойство определителей Вандермонда

$$\begin{vmatrix} 1 & 1 & \ldots & 1 \\ d_1 & d_2 & \ldots & d_k \\ d_1^2 & d_2^2 & \ldots & d_k^2 \\ \vdots & & & \\ d_1^{k-1} & d_2^{k-1} & \ldots & d_k^{k-1} \end{vmatrix}$$

быть отличными от нуля, если все $d_1, d_2, \ldots, d_k$ попарно различны.

В начале работы вероятностная машина выбирает к случайных чисел $\tau_1, \tau_2, \ldots, \tau_k$, каждое из которых независимо друг от друга равновероятно принимает значения из множества $\{1,2,3,\ldots,2^l\}$, где l - наименьшее такое натуральное число, что $2^l > \frac{k}{\varepsilon}$.

Вероятностная машина при чтении каждой буквы a_i $(i \in \{1,2,\ldots,k\})$ прибавляет счетчику число τ_i, а при чтении буквы b_i $(i \in \{1,2,\ldots,k\})$ отнимает число τ_i. Параллельно с помощью детерминированного конечного автомата проверяется имеет ли слово вид $a_1^{n_1} a_2^{n_2} \ldots a_k^{n_k} b_1^{m_1} b_2^{m_2} \ldots b_k^{m_k}$.

Слово принимается, если счетчик пуст и отвергается в противном случае.

Легко видеть, что если слово принадлежит языку L_k, то, независимо от выбора числа τ слово принимается. Если же слово не принадлежит языку L_k, но имеет вид $a_1^{n_1} a_2^{n_2} \ldots a_k^{n_k} b_1^{m_1} b_2^{m_2} \ldots b_k^{m_k}$, то оно принимается тогда и только тогда, когда

$$(n_1 - m_1) \cdot 1 + (n_2 - m_2) \cdot \tau + (n_3 - m_3) \tau^2 + \ldots + (n_k - m_k) \cdot \tau^{k-1} = 0$$

Но такое соотношение может выполняться не более чем для k-1 различных τ : ведь если бы оно выполнялось для k различных $\tau_1, \ldots, \tau_k$, то мы бы имели систему линейных уравнений с неизвестными $(n_1 - m_1), (n_2 - m_2), \ldots, (n_k - m_k)$, определитель которой отличен от нуля. Такая система могла бы иметь только тривиальное решение

$$n_1 - m_1 = n_2 - m_2 = \ldots = n_k - m_k = 0. \qquad \square$$

ПРЕДЛОЖЕНИЕ 2. Для любого $k \in \mathbb{N}^+$ свойство продолжаемости слов до языка L_k может распознаваться на вероятностной односторонней счетчиковой машине с вероятностью $\frac{k}{2k-1}$.

ДОКАЗАТЕЛЬСТВО. В начале работы вероятностная машина равновероятно выбирает одно число $\tau \in \{1,2,3,\ldots,2k-1\}$ может иметь любой конечный алфавит. Если этого не допускать, то надо поступать так. Надо выбрать число, равное полной степени двойки и превышающее 2k-1. Если выпадает

$r \in \{1,2,\ldots,2\kappa-1\}$, то оно и выдается в качестве результата датчика. Если выпадает число из $\{2\kappa,\ldots,2^{\ell}\}$, то результат не выдается и выбор случайного числа производится еще раз.

Если $r \in \{\kappa+1,\kappa+2,\ldots,2\kappa-1\}$, то вероятностная машина выдает результат "слово не может быть продолжено до слова из языка". Если $r \in \{1,2,\ldots,\kappa\}$, то машина параллельно: 1) проверяет как детерминированный конечный автомат, имеет ли входное слово вид $a_1^{n_1}a_2^{n_2}\ldots a_\kappa^{n_\kappa}b_1^{m_1}b_2^{m_2}\ldots b_\kappa^{m_\kappa}$; 2) игнорируя остальные буквы входного слова, при проявлении каждой буквы a_i прибавляет счетчику число +1, и при появлении каждой буквы b_i вычитает из счетчика 1. Слово принимается, если в счетчике находится неотрицательное число.

Если входное слово продолжаемо до слова из языка L_κ, то при $r \in \{1,2,\ldots,\kappa\}$ (т.е. с вероятностью $\frac{\kappa}{2\kappa-1}$) получается правильный результат "продолжаемо".

Если входное слово не продолжаемо до слова из языка L_κ потому, что не продолжаемо уже до слова вида $a_1^{n_1}a_2^{n_2}\ldots a_\kappa^{n_\kappa}b_1^{m_1}b_2^{m_2}\ldots b_\kappa^{m_\kappa}$, то это машина констатирует при помощи вычисления 1) и результат "не продолжаемо" выдает гарантированс.

Если входное слово является начальным фрагментом некоторого слова $a_1^{n_1}a_2^{n_2}\ldots a_\kappa^{n_\kappa}b_1^{m_1}b_2^{m_2}\ldots b_\kappa^{m_\kappa}$, но все же не продолжаемо до слова из языка L_κ, то это означает, что некоторое m_i больше чем соответствующее n_i. Тогда правильный результат "не продолжаемо" машина выдает при $r \in \{i,\kappa+1,\kappa+2,\ldots,2\kappa-1\}$, т.е. при к различных значениях из 2к–1.

ТЕОРЕМА 3. Для любого $\kappa \in N^+$ свойство продолжаемости слов до языка L_κ не может распознаваться на вероятностной односторонней счетчиковой машине с вероятностью, превышающей $\frac{\kappa}{2\kappa-1}$.

ДОКАЗАТЕЛЬСТВО: Допустим от противного, что такая машина M существует, и она распознает продолжаемость слов до языка L_κ с вероятностью $\frac{\kappa}{2\kappa-1}+\Delta$ $(\Delta > 0)$. Дальнейшая

часть доказательства состоит из ряда лемм.

ЛЕММА 3.1. Для любого набора натуральных чисел $(n_2, n_3, \ldots, n_\kappa)$ существует такая константа $c>0$ что для бесконечно многих слов $a_1^{n_1}a_2^{n_2}a_3^{n_3}\ldots a_\kappa^{n_\kappa}b_1^{n_1}$ суммарная вероятность таких конфигураций машины M в момент чтения следующей буквы за этим словом, что число, записанное в счетчике, по абсолютному значению не превосходит c, больше чем $\frac{1}{2\kappa-1}$.

ДОКАЗАТЕЛЬСТВО. Как и всю теорему, лемму мы тоже докажем от противного. Допустим, что для некоторого набора натуральных чисел $(n_2, n_3, \ldots, n_\kappa)$ для всех констант $c>0$ для всех достаточно больших n_1 суммарная вероятность таких конфигураций машины M при работе на слове $a_1^{n_1}a_2^{n_2}a_3^{n_3}\ldots a_\kappa^{n_\kappa}b_1^{n_1}$ в момент чтения следующей буквы за этим словом, что число, записанное в счетчике по абсолютному значению не превосходит c, не больше чем $\frac{1}{2\kappa-1}$.

Обозначим число внутренних состояний машины M через ℓ. М.Рабином [4] доказана верхняя оценка $S(\varepsilon)$ числа точек в ℓ-мерном единичном кубе, таких что никакие две точки не находятся ближе чем на расстоянии ε, где расстояние ρ между точками с координатами $(x_1', x_2', \ldots, x_\ell')$ и $(x_1'', x_2'', \ldots, x_\ell'')$ определяется так:

$$\rho(x', x'') = \sum_{i=1}^{\ell} |x_i'' - x_i'|$$

Для нас не важен конкретный вид функции $S(\varepsilon)$. Важно, что для любого $\varepsilon>0$ значение $S(\varepsilon)$ является конечным.

Пусть c - настолько большое число, что из любой конфигурации, где в счетчике находится число, превосходящее c, вероятность того, что за $4S$ шагов содержание счетчика хотя бы в какой-то момент изменится более чем на c по сравнению с первоначальным, не превосходит δ. (Таким образом, c зависит как от ε, так и от δ.)

Пусть ε и δ - достаточно малые числа (позже в доказательстве этой леммы будет сказано, насколько именно малыми они должны быть).

Рассмотрим распределение вероятностей внутренних состояний машины M при обработке слов

$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1-45},$$
$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1-45+1},$$
$$\dots\dots\dots\dots\dots\dots\dots\dots$$
$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1+}$$

в момент чтения следующей буквы за этим словом. Всего здесь 45 слов, следовательно по оценке М.О.Рабина среди них должна быть такая пара

$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1-U}, \qquad (1)$$

$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1-V}, \qquad (2)$$

что векторы указанных вероятностей для этих слов

$$(x_1', x_2', \dots, x_\ell')$$

$$(x_1'', x_2'', \dots, x_\ell'')$$

в метрике ρ ближе чем на расстояние ε.

Для определенности считаем, что $V > U$. Тогда слово

$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1-U+V}$$

не продолжаемо до слова из языка L_k, а слово

$$a_1^{n_1} a_2^{n_2} a_3^{n_3} \dots a_k^{n_k} b_1^{n_1-V+V}$$

продолжаемо.

Рассмотрим теперь другое распределение вероятностей, а именно - распределение вероятностей конфигураций машины в момент чтения следующей буквы за этим словом. Обозначим эти распределения для слов (1) и (2), соответственно, через

$$(\xi_1', \xi_2', \dots),$$
$$(\xi_1'', \xi_2'', \dots).$$

Заметим три обстоятельства. Во-первых, если конфигурация № i такова, что в счетчике число, превышающее по абсолютному значению число c, и вероятность события "на этом слове машина выдает результат "да" при условии, что не происходит очищения счетчика во время этого вычисления, равна y_i, то и для любой конфигурации № j с тем же внутренним состоянием, но с другим числом в счетчике (это число тоже $>$ c) указанная условная вероятность тоже равна y_i. [Другими словами, если счетчик не очищается, то машина не чувствует, это большое число или маленькое].

Во-вторых, по допущению от противного в начале доказательства леммы суммарная вероятность $\Sigma \xi_i$ по конфигурациям № i с маленьким значением счетчика (не больше c по абсолютному значению) не больше чем $\frac{1}{2\kappa-1}$.

В-третьих, по выбору S за время обработки $4S$ букв b содержание счетчика может очиститься с вероятностью не больше чем δ.

Пусть η_i обозначает вероятность выдачи результата"да" из конфигурации № i без очищения счетчика и ζ_i обозначает вероятность выдачи результата "да" из конфигурации № i с очищением счетчика.

Тогда

$$\xi_1'(\eta_1+\zeta_1)+\xi_2'(\eta_2+\zeta_2)+\ldots < \frac{\kappa-1}{2\kappa-1}-\Delta$$

$$\xi_1''(\eta_1+\zeta_1)+\xi_2''(\eta_2+\zeta_2)+\ldots < \frac{\kappa}{2\kappa-1}+\Delta$$

Первое неравенство остается в силе, если мы:

а) отбрасываем все ζ_i;

б) отбрасываем все конфигурации с содержанием счетчика по абсолютному значению меньше c;

в) учитывая, что $x_j' = \Sigma \xi_i'$ по всем конфигурациям № i с внутренним состоянием j, а также то, что все соответствующие η_i равны (из-за "во-первых"), собираем подобные члены

$$x_1' y_1 + x_2' y_2 + \ldots + x_\ell' y_\ell < \frac{\kappa-1}{2\kappa-1} - \Delta \qquad (3)$$

Второе неравенство остается в силе, если мы:

а) отбрасываем все ξ_i, одновременно уменьшая правую часть неравенства на δ (здесь используется "в-третьих");

б) отбрасываем все конфигурации с содержанием счетчика по абсолютному значению меньше c, одновременно уменьшая правую часть неравенства на $\frac{1}{2\kappa-1}$ (здесь используется "во-вторых");

в) учитывая, что $x_j'' = \sum \xi_i''$ по всем конфигурациям № i с внутренним состоянием j, а также то, что все соответствующие η_i равны (из-за "во-первых"), собираем подобные члены:

$$x_1'' y_1 + x_2'' y_2 + \ldots + x_\ell'' y_\ell > \frac{\kappa-1}{2\kappa-1} + \Delta - \delta \qquad (4)$$

Вычитаем из неравенства (4) неравенство (3):

$$(x_1'' - x_1')y_1 + (x_2'' - x_2')y_2 + \ldots + (x_\ell'' - x_\ell')y_\ell > 2\Delta - \delta$$

Это неравенство остается в силе, если все $y_1, \ldots, y_\ell$ заменить на I (ведь все $y_j \le 1$):

$$(x_1'' - x_1') + (x_2'' - x_2') + \ldots + (x_\ell'' - x_\ell') > 2\Delta - \delta,$$

а также заменить все разности симметрическими разностями

$$|x_1'' - x_1'| + |x_2'' - x_2'| + \ldots + |x_\ell'' - x_\ell'| > 2\Delta - \delta.$$

Теперь мы указываем явно, что δ нами выбрано равным Δ, а ε выбрано равным $\frac{\Delta}{3}$. Последнее неравенство можно переписать в виде

$$\rho(x', x'') > 2\varepsilon.$$

Противоречие с выбором x' и x''.

Докажем теперь следующее обобщение леммы **3.1.**

ЛЕММА 3.2. Для любого $i \in \{1,2,\dots,\kappa\}$ и для любого набора натуральных чисел $(n_{i+1}, n_{i+2}, \dots, n_\kappa)$ существует такая константа $c>0$ и существует такое бесконечное множество слов из языка L_κ, что у них попарно различны компоненты $b_i^{n_i}$ и для всех этих слов суммарная вероятность таких реализаций работы машины, что либо в момент чтения первой буквы b_2, либо в момент чтения первой буквы $b_3, \dots,$, либо в момент чтения первой буквы b_i число, записанное в счетчик, по абсолютному значению не превышает c, больше чем $\frac{i}{2\kappa-1}$.

Прежде чем приступить к доказательству этой леммы, отметим

ЧАСТНЫЙ СЛУЧАЙ ЛЕММЫ 3.2. Существует такая константа $c>0$ и существует такое бесконечное множество слов из языка L_κ, что у них попарно различны компоненты $b_\kappa^{n_\kappa}$ и для всех этих слов суммарная вероятность таких реализаций работы машины, что либо в момент чтения первой буквы b_2, либо в момент чтения первой буквы $b_3, \dots$, либо в момент чтения первой буквы b_κ число, записанное в счетчик, по абсолютному значению не превышает c, больше чем $\frac{\kappa}{2\kappa-1}$.

ДОКАЗАТЕЛЬСТВО ЛЕММЫ 3.2.

Базис индукции. При $i = 1$ лемма доказана (см.лемму 3.1).

Шаг индукции. Пусть лемма доказана для $i-1$. Докажем ее при i .

Допустим от противного, что для некоторого набора натуральных чисел $(n_{i+1}, n_{i+2}, \dots, n_\kappa)$ для всех констант $c>0$ для любого множества $\mathcal{L}$ слов языка L_κ, такого, что у всех слов этого множества

А) компоненты $a_{i+1}^{n_{i+1}} a_{i+2}^{n_{i+2}} \dots a_\kappa^{n_\kappa}$ имеют параметры именно из указанного выше набора,

Б) компоненты $a_i^{n_i}$ и $b_i^{n_i}$ имеют (общую) длину равную любому достаточно большому числу, и эти длины различны для различных рассматриваемых слов, - для всех таких слов суммарная вероятность таких реализаций работы машины, что либо в момент чтения первой буквы b_2 , либо в момент чтения первой буквы $b_3, \dots$, либо в момент чтения первой буквы b_i

число, записанное в счетчик, по абсолютному значению не превышает c, не больше чем $\frac{c}{2k-1}$.

Пусть, как и в доказательстве леммы 3.I, число внутренних состояний машины M обозначено через ℓ . Зафиксируем два "малых" числа ε и δ (ниже в доказательстве будет сказано, насколько малыми они должны быть).

По предположению индукции лемма 3.2 для i -I уже доказана. Пользуясь этим, мы будем дальше рассматривать не произвольные множества $\mathcal{K}$ со свойствами А) и Б), а только такие, где компонента n_{i-1} подобрана исходя из $n_i, n_{i+1}, \ldots, n_k$ по лемме 3.2 (для i -I), компонента n_{i-2} подобрана по $n_{i-1}, n_i, n_{i+1}, \ldots, n_k$ по лемме 3.2 (для i-2), ..., компонента n_1 подобрана по $n_2, \ldots, n_i, n_{i+1}, \ldots, n_k$ по лемме 3.2 (для $i = I$).

Вводим метрику ρ в ℓ -мерном пространстве так же как в лемме 3.I. Точно таким же образом определяем и $S(\varepsilon)$. При определении c появляется еще одно условие (кроме того что уже было в лемме 3.I), а именно: c должно быть не меньше чем c в формулировке леммы 3.2 для i -I.

Будем называть характеристикой [можно любое другое название] реализации работы M на слове из языка L_k следующую информацию о слове и работе машины M на нем: I) если в момент чтения первой буквы b_2 машина имела в счетчике число по абсолютному значению не больше c, то характеристикой является эта конфигурация и сигнал (1) ; 2) если условие I) не выполняется, но в момент чтения первой буквы b_3 машина имела в счетчике число по абсолютному значению не больше c, то характеристикой является эта конфигурация и сигнал (2); ...; i) если условия I), ..., i-I) не выполняются, но в момент чтения первой буквы b_{i+1} машина имела в счетчике число по абсолютному значению не больше c, то характеристикой является эта конфигурация и сигнал (i); $i+1$) если условия I), ..., i) не выполняются, то характеристикой является конфигурация в момент чтения [той же самой] первой буквы b_{i+1} и сигнал ($i+1$) .

Итак, теперь мы рассматриваем распределение вероятно-

стей не конфигураций в какой-то момент, а характеристик.

Заметим, что характеристик с сигналами ①, ②, ..., (i) - только конечное число. Пусть все бесконечно многие характеристики пронумерованы так, что сначала идут все характеристики с сигналами ①, ②, ..., $(i-1)$ (их число обозначим через d - значит они имеют номера от 1 до d), потом - другие характеристики.

Каждому слову из языка L_K будем соотносить точку в $(l+d)$-мерном пространстве. Каждой размерности будут соответствовать номер характеристики (если номер не превышает d) и номер внутреннего состояния (иначе). Точка в $(l+d)$-мерном единичном кубе указывает вероятности этих характеристик, а при характеристиках с сигналами (i) - вероятность внутреннего состояния.

Метрика ρ вводится похожим образом

$$\rho(x', x'') = \sum_{j=1}^{d} |z_j'' - z_j'| + \sum_{j=1}^{l} |x_j'' - x_j'|,$$

где z_j - вероятность характеристик № j, а x_j - вероятность внутреннего состояния j при сигнале (i).

$S(\varepsilon)$ равно некоторой верхней оценке числа точек в $(l+d)$-мерном единичном кубе, где все точки полярно находятся дальше чем ε.

При определении c появляется еще одно условие (кроме того, что уже было в доказательстве леммы 3.1), а именно: c должно быть не меньше чем c в формулировке леммы 3.2 для $i-1$.

Заметим три обстоятельства. Во-первых, если характеристика № j такова, что сигналом является (i), то условная вероятность события "на том слове машина выдает результат "да" при условии, что не происходит очищение счетчика до конца вычисления, равна y_j, то и для любой другой характеристики с тем же внутренним состоянием (в конфигурации) и сигналом (i), но с другим числом в счетчике, указанная условная вероятность тоже равна y_j.

Заметим по допущению от противного в начале доказательства леммы 3.2 суммарная вероятность характеристик с сигналами ①, ②, ..., (i) не больше чем $\frac{i}{2^{K-1}}$. С другой

стороны, из леммы 3.2 для i -I вытекает, что суммарная вероятность характеристик с сигналами ①, ②, ..., $(i-1)$ больше чем $\frac{i-1}{2\kappa-1}$. Поэтому, во-вторых, суммарная вероятность характеристик с сигналом (i) не больше чем $\frac{1}{2\kappa-1}$.

В-третьих, по выбору S за время обработки $4S$ букв b_i содержание счетчика может очиститься с вероятностью не больше чем δ .

Рассмотрим распределение вероятностей внутренних состояний машины M при обработке слов

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i - 4S},$$

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i - 4S+1},$$

. . . . . .

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i}$$

в момент чтения следующей буквы за этим словом. Всего здесь $4S$ слов, следовательно, по оценке М.О.Рабина среди них должна быть такая пара

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i - U}, \qquad (2.1)$$

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i - V}, \qquad (2.2)$$

что векторы указанных вероятностей для этих слов

$$(x_1', x_2', \dots, x_\ell')$$

$$(x_1'', x_2'', \dots, x_\ell'')$$

в метрике ρ ближе чем на расстояние ϵ .

Для определенности считаем, что $V > U$. Тогда прибавляя V букв b_i получаем слово

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i - U - V},$$

которое не продолжаемо до слова и языка L_κ , и слово

$$a_1^{n_1} a_2^{n_2} \dots a_\kappa^{n_\kappa} b_1^{n_1} \dots b_i^{n_i - V + V}$$

которое продолжаемо.

Пусть η_j обозначает вероятность выдачи результата "да" из характеристики № j без очищения счетчика и ζ_j обозначает вероятность выдачи результата "да" из характеристики № j с очищением счетчика.

Тогда

$$\xi_1'(\eta_1+\zeta_1)+\xi_2'(\eta_2+\zeta_2)+\ldots<\frac{\kappa-1}{2\kappa-1}-\Delta,$$

$$\xi_1''(\eta_1+\zeta_1)+\xi_2''(\eta_2+\zeta_2)+\ldots>\frac{\kappa}{2\kappa-1}+\Delta.$$

Первое неравенство остается в силе, если мы:

а) отбрасываем все ζ_i;

б) отбрасываем все характеристики с сигналом ⓘ;

в) учитывая, что $x_t'=\sum\xi_j'$ по всем характеристикам № j с внутренним состоянием t и сигналом $(i+1)$, а также то, что все соответствующие η_j равны (из-за "во-первых"), собираем подобные члены

$$x_1'\cdot y_1+x_2'\cdot y_2+\ldots+x_t'\cdot y_t+x_{t'}'\cdot\eta_1+\ldots+x_d'\cdot\eta_d<\frac{\kappa-1}{2\kappa-1}-\Delta \quad (2.3)$$

Второе неравенство остается в силе, если мы:

а) отбрасываем все ζ_i, одновременно уменьшая правую часть неравенства на δ (здесь используется "в-третьих");

б) отбрасываем все характеристики с сигналом ⓘ, одновременно уменьшая правую часть неравенства на $\frac{1}{2\kappa-1}$ (здесь используется "во-вторых");

в) учитывая, что $x_t''=\sum\xi_j''$ по всем характеристикам № j с внутренним состоянием t и сигналом $(i+1)$, а также то, что все соответствующие η_j равны (из-за "во-первых"), собираем подобные члены:

$$x_1''y_1+x_2''y_2+\ldots+x_t''y_t+x_{t'}''\eta_1+\ldots+x_d''\eta_d>\frac{\kappa-1}{2\kappa-1}+\Delta-\delta \quad (2.4)$$

Вычитаем из неравенства (2.4) неравенство (2.3):

$$(x_1''-x_1')y_1+(x_2''-x_2')y_2+\ldots+(x_\ell''-x_\ell')y_\ell+$$
$$+(z_1''-z_1')\eta_1+\ldots+(z_\delta''-z_\delta')\eta_\delta > 2\Delta-\delta.$$

Это неравенство остается в силе, если все $y_1,\ldots,y_\ell$, $\eta_1,\ldots,\eta_\delta$ заменить на 1, и потом заменить все разности симметрическими разностями:

$$|x_1''-x_1'|+\ldots+|x_\ell''-x_\ell'|+|z_1''-z_1'|+\ldots+|z_\delta''-z_\delta'|>2\Delta-\delta.$$

Теперь мы указываем, что δ нами выбрано равным Δ, а ε выбрано равным $\frac{\Delta}{3}$. Последнее неравенство можно переписать в виде

$$\rho(x',x'') > \frac{2\Delta}{3} = 2\varepsilon$$

Противоречие с выбором x' и x''.

ДОКАЗАТЕЛЬСТВО ТЕОРЕМЫ 3. Итак, выполняется частный случай леммы 3.2. Для каждого из слов бесконечного множества, указанного в формулировке этого частного случая мы рассматриваем распределение вероятностей конфигураций в момент чтения первой буквы b_k.

Пусть e - верхняя оценка числа конфигураций в этот момент, где в счетчике число, по абсолютному значению не превышающее с. Соотнесем каждому слову из множества точку в e-мерном единичном кубе, означающую вероятность этой конфигурации.

Так как точек бесконечно много, то должны найтись 2 точки, для которых расстояние в смысле метрики

$$\rho(x',x'')=\sum_{j=1}^{e}|x_j''-x_j'|$$

меньше чем ε.

Итак, пусть эти точки соответствуют словам

$$a_1^{n_1}a_2^{n_2}\ldots a_{k-1}^{n_{k-1}}a_k^{n_k}b_1^{n_1}b_2^{n_2}\ldots b_{k-1}^{n_{k-1}},$$
$$a_1^{m_1}a_2^{m_2}\ldots a_{k-1}^{m_{k-1}}a_k^{m_k}b_1^{m_1}b_2^{m_2}\ldots b_{k-1}^{m_{k-1}},$$

причем $m_k > n_k$.

Рассмотрим слова

$$a_1^{n_1} a_2^{n_2} \dots a_{k-1}^{n_{k-1}} a_k^{n_k} b_1^{n_1} b_2^{n_2} \dots b_{k-1}^{n_{k-1}} b_k^{n_k},$$

$$a_1^{m_1} a_2^{m_2} \dots a_{k-1}^{m_{k-1}} a_k^{m_k} b_1^{m_1} b_2^{m_2} \dots b_{k-1}^{m_{k-1}} b_k^{m_k}.$$

Первое из этих слов принадлежит языку L_k , а второе нельзя продолжить до слова из языка L_k .

Пусть через y_j обозначена вероятность выдать "да", исходя из конфигурации № j .

$$x_1' y_1 + x_2' y_2 + \dots > \frac{k}{2k-1} + \Delta .$$

Отбросим компоненты, соответствующие конфигурациям с большим значениям счетчика и одновременно вычтем из правой части неравенства $\frac{1}{2k-1}$.

Неравенство остается в силе

$$x_1' y_1 + x_2' y_2 + \dots + x_\ell' y_\ell > \frac{k-1}{2k-1} + \Delta$$

Для другого слова

$$x_1'' y_1 + x_2'' y_2 + \dots + x_\ell'' y_\ell < \frac{k-1}{2k-1} - \Delta$$

Вычитая получаем

$$(x_1'' - x_1') y_1 + \dots + (x_\ell'' - x_\ell') y_\ell > 2\Delta$$

$$|x_1'' - x_1'| + \dots + |x_\ell'' - x_\ell'| > 2\Delta$$

Если ε было взято меньше чем Δ , то противоречие. □

Наконец, без доказательства отметим следующую теорему, доказательство которой (из-за большого объема текста) будет опубликовано в другом месте.

ТЕОРЕМА 4. Существует язык L_0, для которого: 1) при любом $\varepsilon > 0$ язык L_0 можно распознавать на вероятностной односторонней счетчиковой машине с вероятностью $1-\varepsilon$; 2) свойство продолжаемости слов до языка L_0 алгоритмически неразрешимо.

Таким образом доказано, что хотя сам язык L распознаваем на вероятностных односторонних счетчиковых машинах со сколь угодно высокой вероятностью $1-\varepsilon$ $(\varepsilon > 0)$, но свойство продолжаемости слов до языка L может быть распознаваемо на вероятностных односторонних счетчиковых машинах с вероятностью 2/3 и не выше, может быть распознаваемо на вероятностных односторонних счетчиковых машинах с вероятностью 3/5 и не выше, ..., может быть распознаваемо на вероятностных односторонних счетчиковых машинах с вероятностью $к/2к-1$ и не выше, ..., может быть нераспознаваемо. Хотелось бы знать, существует ли такое число р, что $p \notin \{ к/2к-1 \}_{к=2,3,\ldots}$, и такой язык L, что сам L распознаваем на вероятностных односторонних счетчиковых машинах со сколь угодно высокой вероятностью $1-\varepsilon$ $(\varepsilon > 0)$ а свойство продолжаемости слов до языка L может быть распознаваемо на вероятностных односторонних счетчиковых машинах с вероятностью р и не выше.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Hopcroft J.E., Ullman J.D. Formal languages and their relation to automata // New-York et al.: Addison-Wesley, 1969.
2. Chytil M.P., Demner J. Calmly on panic mode // Lecture Notes in Computer Science. Berlin et al.: Springer. 1987. P.267.
3. Фрейвалд Р.В. Возможности различных моделей односторонних вероятностных автоматов // Известия ВУЗ. Сер. Математика, 1981. № 5(228). С.26-34.
4. Рабин М.О. Вероятностные автоматы // Кибернетический сборник. М.: Мир, 1964. Вып.9. С.123-141

ИНДУКТИВНЫЙ СИНТЕЗ ОБЩИХ ГРАФИЧЕСКИХ ВЫРАЖЕНИЙ

И.Этмане
ВЦ при ЛГУ им.П.Стучки

Разъясняя различные алгоритмы, человек часто пользуется примерами. Например, алгоритм решения системы линейных уравнений методом Гаусса можно описать с помощью примера, изображенного на рис.2 (здесь выход по свободной дуге интерпретируется как конец примера).

Очевидно, имеет смысл попытаться формализовать методы и языки, которыми пользуется человек, объясняя алгоритмы таким образом. Одной из возможных формализаций является т.н. графические выражения (г.в.), рассмотренные в [I]. В [I] г.в. были рассмотрены с прагматической точки зрения. В данной работе дается дальнейшая формализация г.в., формируется задача синтеза и приводится теорема о существовании алгоритма, решающего задачу синтеза.

Данная работа является развитием теоретических исследований, начатых в [2] (там был рассмотрен линейный случай г.в.).

Под элементарным структурным графом (э.с.г.) мы будем понимать любой из графов, изображенных на рис. Iа и Iб.

Рис.Iа

Рис.Iб

Таким образом, э.с.г. - это граф с одной вершиной и с одной или с двумя свободными дугами (при этом, если две дуги, то одна из них помечена символом '+', а другая - символом '-').

В дальнейшем под структурными графами (с.г.) мы будем понимать следующие графы.

I. Э.с.г. является с.г.

2. Пусть G - с.г. Позволим выполнить в графе G .

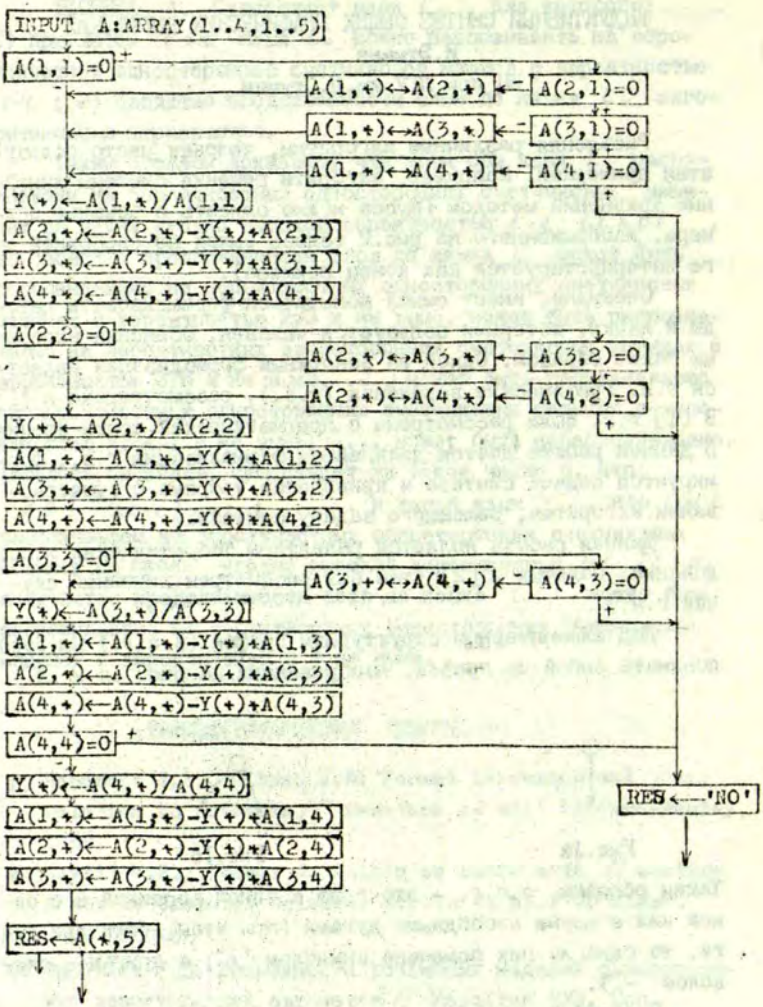

INPUT A:ARRAY(1..4,1..5)
A(1,1)=0
A(1,*)↔A(2,*)
A(2,1)=0
A(1,*)↔A(3,*)
A(3,1)=0
A(1,*)↔A(4,*)
A(4,1)=0
Y(*)←A(1,*)/A(1,1)
A(2,*)←A(2,*)-Y(*)*A(2,1)
A(3,*)←A(3,*)-Y(*)*A(3,1)
A(4,*)←A(4,*)-Y(*)*A(4,1)
A(2,2)=0
A(2,*)↔A(3,*)
A(3,2)=0
A(2,*)↔A(4,*)
A(4,2)=0
Y(*)←A(2,*)/A(2,2)
A(1,*)←A(1,*)-Y(*)*A(1,2)
A(3,*)←A(3,*)-Y(*)*A(3,2)
A(4,*)←A(4,*)-Y(*)*A(4,2)
A(3,3)=0
A(3,*)↔A(4,*)
A(4,3)=0
Y(*)←A(3,*)/A(3,3)
A(1,*)←A(1,*)-Y(*)*A(1,3)
A(2,*)←A(2,*)-Y(*)*A(2,3)
A(4,*)←A(4,*)-Y(*)*A(4,3)
A(4,4)=0
Y(*)←A(4,*)/A(4,4)
A(1,*)←A(1,*)-Y(*)*A(1,4)
A(2,*)←A(2,*)-Y(*)*A(2,4)
A(3,*)←A(3,*)-Y(*)*A(3,4)
RES←A(*,5)
RES←'NO'

Рис.2.

подстановки, показанные на рис.3а, 3б, 3в.

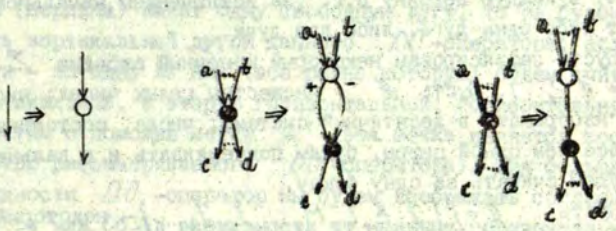


Рис.3а Рис.3б Рис.3в

Граф G', полученный из G, применением один или несколько раз показанных выше подставок, является с.г.

Например, графы, изображенные на рис.4а и 4б, являются с.г.

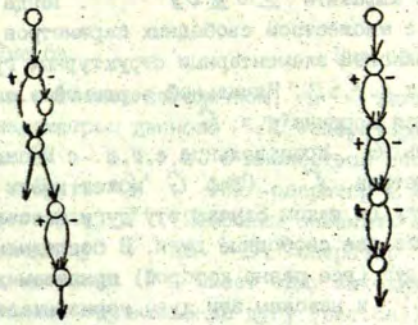


Рис.4а Рис.4б

Отметим, что определенные выше с.г. обладают следующими свойствами:

1) с.г. содержит только одну вершину, в которую не входят дуги; такую вершину в дальнейшем будем называть начальной;

2) с.г. имеет либо одну свободную дугу, либо две свободные дуги;

3) из каждой вершины с.г. выходят либо одна дуга, либо две дуги (помеченные, соответственно, метками '+' и '-');

4) в каждую вершину с.г. (за исключением начальной) входят либо одна дуга, либо две дуги.

Пусть зафиксирован некоторый конечный алфавит Σ (, , $= \notin \Sigma$). Пусть $\mathbb{Z}$ – множество целых чисел; числа будем изображать в десятичной системе; числа, состоящие из более чем одной цифры, будем подчеркивать и в дальнейшем будем считать за одну букву.

Пусть $\mathbb{I} = \{I, J, K, L, M, \dots, I_1, I_2, \dots\}$ – некоторый конечный алфавит т.н. параметров и пусть $\bar{\mathbb{I}} = \{\underline{I+c}, I \in \mathbb{I}, c \in \mathbb{Z}\}$. Для удобства вместо $\underline{I+(-c)}$ будем писать $\underline{I-c}$ а вместо $\underline{I+0}$ – просто $\underline{I}$.

Далее определим структурное графическое выражение (с.г.в.) с заданным множеством свободных параметров $\mathbb{I}'$.

I. Пусть $\mathbb{I}'$ некоторое подмножество $\mathbb{I}$ и $\bar{\mathbb{I}} = \{\underline{I+c}, I \in \mathbb{I}', c \in \mathbb{Z}\}$. Пусть G с.г. и его вершины помечены словами в алфавите $\Sigma \cup \mathbb{Z} \cup \bar{\mathbb{I}}$. Тогда G является с.г.в. с множеством свободных параметров $\mathbb{I}'$. Такое с.г.в. назовем элементарным структурным графическим выражением (э.с.г.в.). Начальной вершиной э.с.г.в. является начальная вершина с.г. G.

2. Пусть G произвольное с.г.в. с множеством свободных параметров $\mathbb{I}'$. Граф G может иметь либо одну свободную дугу (в таком случае эту дугу назовем вертикальной), либо две свободные дуги. В последнем случае одной из этих дуг (все равно которой) припишем дополнительную метку '*' и назовем эту дугу горизонтальной, а вторую – вертикальной. Рассмотрим вершину с содержанием (меткой) в виде пары $(I = \mathcal{L}, \beta; G)$ $(I \in \mathbb{I}', \mathcal{L}, \beta \in \{\mathbb{I}' \setminus \{I\} \cup \mathbb{Z}\})$, имеющую одну свободную дугу, если G имеет только вертикальную дугу, и две свободные дуги, если G имеет как вертикальную, так и горизонтальную дуги. Такую вершину также будем называть с.г.в., а множество ее свободных параметров определим равным $\mathbb{I}' \setminus \{I\}$. Такое с.г.в. будем называть графическим DO -оператором

(DO -оператором). Начальной вершиной DO-оператора будем считать начальную вершину с.г. G . Если DO-оператор (вершина) имеет одну свободную дугу, то ее будем считать вертикальной дугой данного DO-оператора; если две дуги - то одну из них (все равно которую) будем считать вертикальной, а вторую горизонтальной; горизонтальную дугу отметим с помощью метки '$*$' (эта метка является составной частью рассматриваемого DO-оператора). Для бо́льшей наглядности DO -оператор мы будем изображать согласно рис.6а или 6б (в зависимости от наличия горизонтальной дуги).

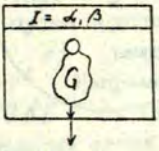


Рис.6а

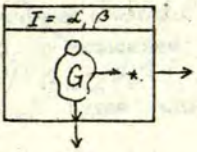


Рис.6б

$I = \alpha, \beta$ будем называть заголовком, G - телом и I - собственным параметром данного DO-оператора.

3. Пусть G - э.с.г.в. с множеством свободных параметров Π' . Пусть G^* - DO-оператор с множеством свободных параметров Π'. Позволим заменить в графе G произвольную вершину v на DO-оператор G^* согласно рис.7 (если вершина v имеет одну выходящую дугу и G^* имеет только вертикальную дугу (рис.8а)) или согласно рис.8 (если v имеет две выходящие дуги и G^* имеет как вертикальную, так и горизонтальную дугу).

Тогда граф G' , полученный из G путем применения к одной или нескольким вершинам v графа G подстановки описанного выше вида (для разных вершин v подставляемые DO-операторы G^* могут быть разными), является с.г.в. с множеством свободных параметров Π'.

Далее определим развертку с.г.в.

Введем необходимые понятия.

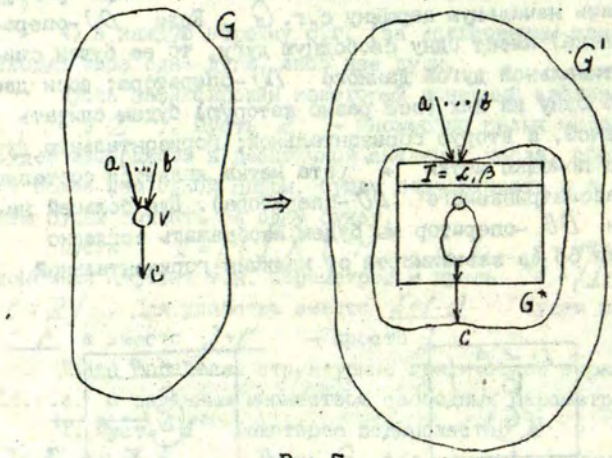


Рис.7.

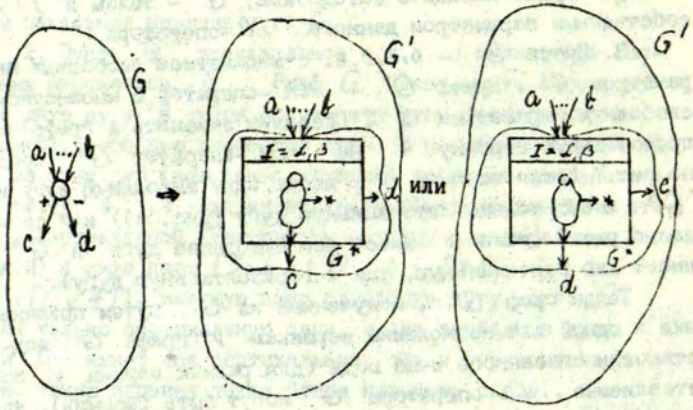


Рис.8.

Определим глубину с.г.в. индуктивно. Глубину э.с.г.в. будем считать равной единице. Глубину DO-оператора будем считать равной $n+1$, где n глубина тела данного DO-оператора. Глубину выражения G определим как максимальную из глубин, входящих в с.г.в. G DO-операторов.

Если множество свободных параметров с.г.в. пустое, то такое с.г.в. будем называть константным.

Далее введем ряд дополнительных обозначений.

Пусть A - некоторое слово в алфавите $\Sigma \cup \mathcal{Z} \cup \mathcal{I}$ и пусть $I_1, \dots, I_k$ символы из алфавита $\mathcal{I}$, входящие в A. Тогда часто будем писать $A(I_1, \dots, I_k)$ и через $A(a_1, \dots, a_k)$ $(a_1, \dots, a_k \in \mathcal{Z})$ будем понимать слово, полученное из A путем подстановки в подчеркнутых символах $\underline{I_i + C}$ $(i \in 1, \dots, k)$ вместо I_i числа a_i и вычисляем значения $a_i + C$. Например, если $A(I,J) = a\underline{I+1}b\underline{J+3}$, то $A(2,3) = a3bC$. Слово $A(a_1, \dots, a_k)$ будем называть конкретизацией слова $A(I_1, \dots, I_k)$.

Пусть G - с.г.в. с множеством свободных параметров $I_1, \dots, I_k$. Тогда будем писать $G(I_1, \dots, I_k)$. Под конкретизацией $G(a_1, \dots, a_k)$ с.г.в. $G(I_1, \dots I_k)$ при заданных $a_1, \dots, a_k$ будем понимать с.г.в. G, в котором заменены все метки-слова (которые мы получаем, спускаясь по глубине DO-операторов) их конкретизациями.

Пусть $I = \alpha, \beta$ - заголовок DO-оператора $\bar{T}$ и пусть $\alpha, \beta \in \mathcal{Z}$. α, β будем называть границами. Тогда число

$$\delta = \begin{cases} +1, & \text{если } \alpha \le \beta, \\ -1, & \text{если } \alpha > \beta \end{cases}$$

назовем направлением DO-оператора $\bar{T}$.

Будем говорить, что DO-оператор $\bar{T}_1$ находится внутри DO-оператора $\bar{T}_2$, если тело $\bar{T}_1$ входит в тело DO-оператора $\bar{T}_2$. В дальнейшем обозначение $\bar{T} = \langle T \rangle_{I=\alpha,\beta}$ мы будем использовать вместо высказывания "$\bar{T}$ является DO-оператором с заголовком $I=\alpha,\beta$, телом T".

Пусть $\bar{T}$ - некоторый внешний DO-оператор, входящий в константное с.г.в. G (в таком случае границы

α, β являются константами), и пусть вертикальная дуга $\bar{T}$ входит в вершину $w_1 \in G$ и горизонтальная дуга (если такая имеется) входит в $w_2 \in G$. Разверткой $\bar{T} = \langle T \rangle_{I=\alpha,\beta}$ в G назовем с.г.в., полученное из G следующим образом:

I. Вычеркиваем из G терм $\bar{T}$.

2. Берем графы $H_0 = T(\alpha)$, $H_1 = T(\alpha+\delta)$, ..., $H_S = T(\beta)$, где

$$\delta = \begin{cases} +1, & \text{если } \alpha \le \beta, \\ -1, & \text{если } \alpha > \beta \end{cases}$$

и вставляем их в G вместо вычеркнутого терма $\bar{T}$, одновременно выполняя следующие подсоединения:

I) дуги, которые в графе G входили в $\bar{T}$, подсоединяем к начальной вершине H_0,

2) вертикальные дуги графа H_i $(i \in 0,1,..,S-1)$ подсоединяем к начальной вершине графа H_{i+1},

3) вертикальную дугу графа H_S подсоединяем к вершине W_1,

4) все горизонтальные дуги графов $H_0, \ldots, H_S$ (если такие имеются) подсоединяем к вершине W_2.

Выражение H, полученное из константного с.г.в. путем последовательной развертки некоторых его DO -операторов, начиная с внешних, назовем частичной разверткой G. Частичную развертку с.г.в. G, не содержащую ни одного DO-оператора (т.е. в которой развернуты все DO-операторы), назовем полной разверткой и обозначим через $Unf(G)$.

Под разверткой с.г.в. $G(I_1, \ldots, I_n)$ при $I_1 = c_1, \ldots, I_n = c_N$ $(c_i \in N)$ будем понимать $Unf(G(c_1, \ldots, c_n))$.

С помощью с.г.в., например, можно записать общий алгоритм решения систем линейных уравнений методом Гаусса, как это показано на рис.9, а изображенный на рис.2 граф (пример работы алгоритма Гаусса) является разверткой данного с.г.в. при $N=4$.

Таким образом, проблема синтеза алгоритмов по примерам их работы можно рассматривать как проблему синтеза с.г.в. по их разверткам (примерам). Конечно, не любой

алгоритм можно записать в виде с.г.в., но как показывает приведенный пример, класс алгоритмов, которые можно записать с помощью с.г.в., охватывает достаточно много содержательных случаев и поэтому изучение класса с.г.в. представляет определенный интерес.

Содержательно с.г.в. с DO-операторами без горизонтальных дуг соответствуют тем, которые в традиционных языках программирования принято называть структурными программами. Развертки таких с.г.в. также не выходят за пределы определения с.г.в. С другой стороны, с.г.в. с DO-операторами, содержащими также горизонтальные дуги, представляют собой обобщение традиционных структурных программ в том смысле, что из тел DO-операторов допускаются также "боковые" выходы. Это существенно расширяет класс рассматриваемых с.г.в. и области их применений. Так, например, рассмотренный выше алгоритм Гаусса естественным образом может быть записан лишь с помощью с.г.в., имеющего DO-операторы с горизонтальными дугами. Но с другой стороны, при развертке с.г.в. с упомянутыми DO-операторами мы, вообще говоря, можем получить графические выражения, которые не принадлежат введенному выше классу с.г.в. (например, приведенный на рис.2 пример). Поэтому теперь несколько расширим понятие с.г.в. с таким расчетом, чтобы их развертки не выводили из рассматриваемого класса графических выражений.

Новый более общий класс графических выражений назовем просто графическими выражениями (г.в.) и его определение от приведенного выше определения класса с.г.в. по существу будет отличаться только в пункте 3. Более точно, определение просто г.в. также состоит из трех пунктов $1'$, $2'$ и $3'$. Пункты $1'$ и $2'$ формулируются в точности так же как пункты 1 и 2 для с.г.в., только вместо "с.г.в." стоит просто "г.в."

3 . Пусть G - э.г.в. с множеством свободных параметров X' . Пусть $G_1, \ldots, G_n$ - г.в. с множествами свободных параметров X' . Каждый из G_i может иметь либо одну свободную дугу, либо две свободные дуги. Если G_i

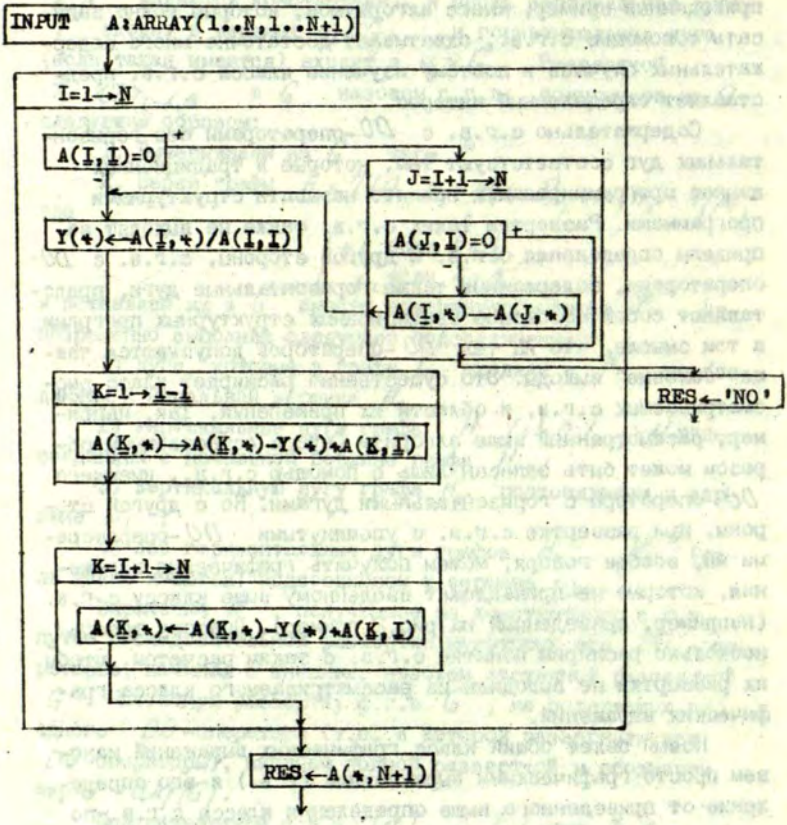

INPUT A:ARRAY(1..N,1..N+1)
I=1→N
A(I,I)=0
+
-
J=I+1→N
Y(*)←A(I,*)/A(I,I)
A(J,I)=0
+
-
A(I,*)—A(J,*)
RES←'NO'
K=1→I-1
A(K,*)→A(K,*)-Y(*)*A(K,I)
K=I+1→N
A(K,*)←A(K,*)-Y(*)*A(K,I)
RES←A(*,N+1)

Рис.9.

имеет две свободные дуги, то выбираем одну из них и называем главной (если G_i имеет только одну свободную дугу, то для единообразия ее также назовем главной). Пусть G^* получен из $G_1, \dots, G_n$ путем подсоединения главной дуги графа G_i $(i \in 1, \dots, n-1)$ к начальной вершине графа G_{i+1}. Начальной вершиной граф. G^* считается начальная вершина графа G_1. Граф G^* обязательно имеет свободную дугу, соответствующую главной свободной дуге графа G_n; эту свободную дугу назовем вертикальной дугой графа G^*. Но в общем случае граф G^* может иметь также и другие свободные дуги, ссответствующие неглавным свободным дугам графов $G_1, \dots, G_n$. Эти дуги для большей наглядности будем рисовать как дуги, сливающиеся в одну дугу (рис.I0) и в дальнейшем

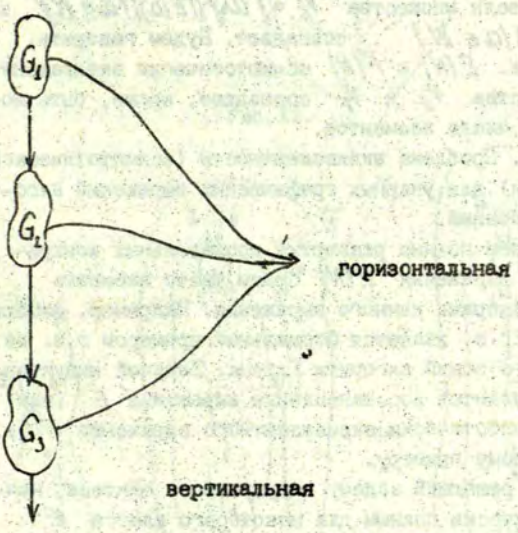


Рис.I0.

эту "результирующую" дугу будем называть горизонтальной дугой графа G^*. Далее, позволим заменить в графе G произвольную вершину v на граф G^*, согласно рис.II (если вершина имеет только одну выходящую дугу и G^* не имеет горизонтальную дугу) или согласно рис.I2 (если v имеет две выходящие дуги и G^* имеет горизонтальную дугу).

Тогда граф G', полученный из G путем применения к одной или нескольким вершинам v графа G подстановки описанного выше вида (для разных v подставляемые графы G^* могут быть разными) является г.в. с множеством свободных параметров $\mathbb{I}'$.

Графические выражения, множество свободных параметров которых состоит только из одного параметра, назовем унарными. Будем говорить, что два унарных г.в. $E(N)$ и $F(N)$ эквивалентны, если множества $V_E = \{Unf(E(a)) \mid a \in \mathbb{N}\}$ и $V_F = \{Unf(F(a)) \mid a \in \mathbb{N}\}$ совпадают. Будем говорить, что унарные г.в. $E(N)$ и $F(N)$ асимптотически эквивалентны, если множества V_E и V_F совпадают, кроме, быть может, конечного числа элементов.

Теорема I. Проблема эквивалентности (асимптотической эквивалентности) для унарных графических выражений алгоритмически разрешима.

В дальнейшем полные развертки произвольных конкретизаций $E(a)$ выражения $E(N)$ будем часто называть формальными примерами данного выражения. Например, изображенное на рис.2г.в. является формальным примером г.в. из рис.9, задающего общий алгоритм Гаусса. Задачей индуктивного синтеза является восстановление выражения E (или выражения, асимптотически эквивалентного выражению E) по его формальному примеру.

Алгоритм, решающий задачу индуктивного синтеза, назовем асимптотически полным для некоторого класса $\mathbb{E}$ г.в., если для любого выражения $F \in \mathbb{E}$ существует такая константа $C_F \in \mathbb{N}$, что для любого $K_0 > C_F$ имеет место следующее: результат применения алгоритма к $Unf(F(K_0))$ является графическим выражением асимптоти-

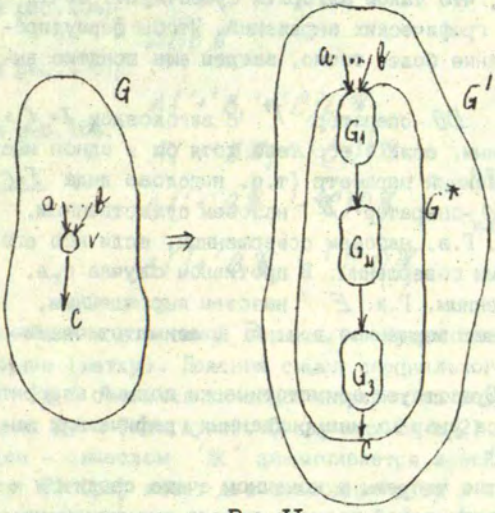


Рис.II.

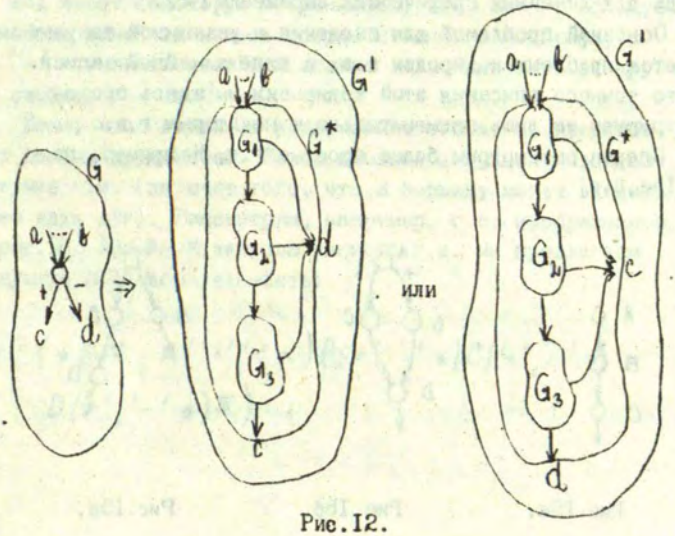


Рис.I2.

чески эквивалентным F .

Оказывается, что такой алгоритм существует "почти" для всех унарных графических выражений. Чтобы формулировать это утверждение более точно, введем еще понятие вырожденности г.в.

Графический $D0$-оператор $\bar{P}$ с заголовком $I=\alpha,\beta$ назовем совершенным, если в его тело хотя бы в одном месте входит его собственный параметр (т.е. подслово вида $I+C$, $C\in\mathbb{Z}$). $D0$-оператор $\bar{P}$ назовем существенным, если $\alpha-\beta\notin\mathbb{Z}$. Г.в. назовем совершенным, если все его существенные термы совершенны. В противном случае г.в. назовем несовершенным. Г.в. E назовем вырожденным, если существует несовершенное г.в. F , асимптотически эквивалентное E .

Теорема 2. Существует асимптотически полный алгоритм синтеза для класса унарных невырожденных графических выражений.

Доказательство теоремы в конечном счете сводится к теореме об асимптотической полноте правил индуктивного вывода для линейных графических выражений [2].

Основной проблемой для сведения к указанной теореме является проблема кодировки г.в. в виде линейной записи. Вместо точного описания этой кодировки мы здесь продемонстрируем ее лишь применительно к некоторым г.в.

Сперва рассмотрим более простые г.в. Например, на рис.I5а,б,в

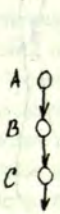


Рис.I5а.

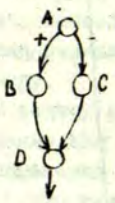


Рис.I5б.

Рис.I5в.

изображенные г.в. линейно мы будем записывать следующим образом:

I) для рис.I5а:

$$ABC\divideontimes$$

2) для рис.I5б.:

$$A('+'B,'-'C)D\divideontimes$$

3) для рис.I5в.:

$$A('+'B\divideontimes,'-'CD)\divideontimes$$

или

$$A('+'B\divideontimes,'-'C)D\divideontimes$$

или

$$A('+'B\divideontimes,'-')CD\divideontimes$$

Под линейным кодом самой вершины мы будем понимать ее содержание (метку). Поясним смысл специального символа $'\divideontimes'$. $'\divideontimes'$ ставится за кодом вершины из которой выходит свободная дуга (или дуги). В случаях I) и 2) его смысл очевиден - символом $'\divideontimes'$ заканчивается линейная запись. В случае 3) граф имеет две свободные дуги и в таком случае линейная запись содержит два символа $'\divideontimes'$, но данный код имеет некоторую особенность, т.е. за $'+'$ следует полностью линейная запись соответственного подграфа, а за $'-'$ - часть линейной записи можно выносить за круглые скобки.

Ясно, что линейное кодирование с.г.в. не представляет особых трудностей. Более сложная ситуация возникает в случае г.в. (за счет того, что в вершину могут входить более двух дуг). Рассмотрим, например, г.в. изображенное на рис.I6. Линейной записью данного г.в. мы предлагаем следующую последовательность:

$$(*E\divideontimes, A('+','-*)B('+','-*)C('+','-*)$$
$$D('+','-*)\divideontimes)$$

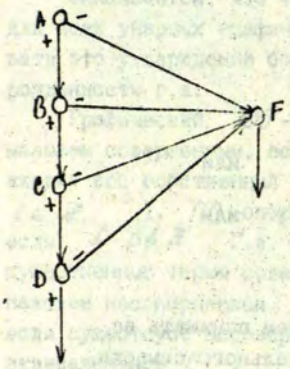


Рис.I6.

Специальный символ '*' здесь ставится перед кодом вершины, в которую входят более двух дуг , и также на место тех дуг, которые входят в эту вершину.

Далее рассмотрим более сложное г.в. - формальный пример алгоритма Гаусса (рис.2). На рис.I7 метки вершин заменены на более короткие и линейный код примера алгоритма Гаусса мы дадим согласно рис.I7. Итак, линейной записью г.в., изображенного на рис.I7 является следующая последовательность:

(*R2*, (*H1 H2 H3 H4, A1('+', '-'*) B1('+', '-'E1*)
B2('+', '-'E2*) B3('+'**, '-'E3*)) (*I1 I2 I3 I4,
A2('+', '-'*) C1('+', '-'F1*) C2('+'**, '-'F2*))
(*J1 J2 J3 J4, A3('+', '-'*) D1('+'**, '-'G1*))
A4('+'*, '-'K1 K2 K3 K4 R1*)).

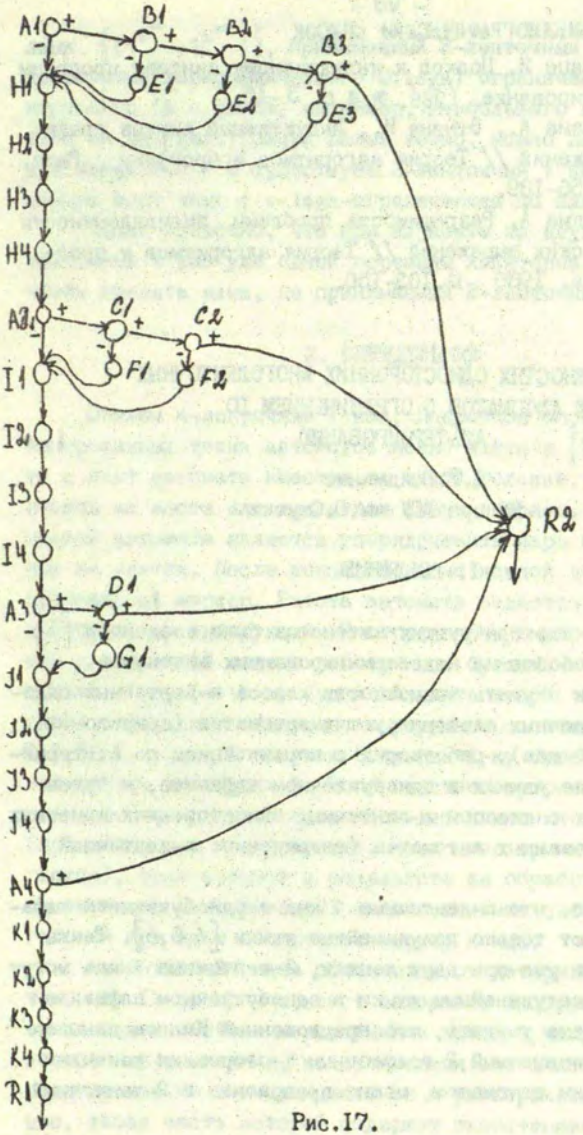

A1
B1
B2
B3
H1
E1
E2
E3
H2
H3
H4
A2
C1
C2
I1
F1
F2
I2
I3
I4
R2
A3
D1
G1
J1
J2
J3
J4
A4
K1
K2
K3
K4
R1

Рис.17.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

I. Этмане И. Подход к индуктивному синтезу программ // Программирование. 1988. № 4.С. 3-16.

2. Бразма А., Этмане И., Индуктивный синтез графических выражений // Теория алгоритмов и программ, Рига, 1986. С.156-189.

3. Бразма А. Разрешимость проблемы эквивалентности для графических выражений // Теория алгоритмов и программ. Рига, 1986 С.103-156.

О ВОЗМОЖНОСТЯХ ОДНОСТОРОННИХ МНОГОЛЕНТОЧНЫХ КОНЕЧНЫХ АВТОМАТОВ С ОГРАНИЧЕНИЕМ ПО АЛЬТЕРНИРОВАНИЮ

Д.Г.Гейдманис
ВЦ при ЛГУ им.П.Стучки

I. ВВЕДЕНИЕ

Понятие альтернирующих автоматов было введено в [I] как сильное обобщение недетерминированных автоматов.

Мы будем изучать возможности класса n-ленточных односторонних конечных альтернирующих автоматов (сокращенно: n-ленточный 1 каа), работающих с ограничением по альтернированию [8] на языках в однобуквенном алфавите, и будем сравнивать их с классом n-ленточных односторонних конечных недетерминированных автоматов (сокращенно: n-ленточный 1 кна).

Известно, что n-ленточные 1 кна в однобуквенном алфавите принимают только полулинейные языки [4,5,6]. Также известно, что уже при двух лентах, 2-ленточные 1 каа могут принимать неполулинейные языки в однобуквенном алфавите [4,7]. Нетрудно увидеть, что предложенный Кингом язык $\{1^{2^m}\}$, принимаемый 2-головочным 1-сторонним конечным альтернирующим автоматом, можно превратить в 2-ленточный

язык $\{(1^{2^m}, 1^{2^m})\}$, принимаемый 2-ленточным 1 каа, притом достаточно логарифмического O(logn) ограничения по альтернированию (n - длина, например, наибольшего из входных слов на лентах). Говоря более точно, можно показать, что для каждого $c > 0$ существует 2-ленточный 1 каа A_c, принимающий этот язык с $c \cdot \log n$-ограничением по альтернированию.

Также известно, что при алфавите из двух букв для 2-ленточных 1 каа уже одной перемены кванторов достаточно, чтобы принять язык, не принимаемый 2-ленточным 1 кна [3].

2. ОПРЕДЕЛЕНИЯ

Опишем n-ленточные 1 каа. Подробное определение детерминированных таких автоматов можно найти в [9]. На каждой из n лент автомата имеется по одной головке. Головки могут стоять на месте или двигаться слева направо. Входной информацией автомата является упорядоченная пара слов, написанных на лентах. После конца слова на каждой ленте написан специальный маркер. Работа автомата задается программой, состоящей из инструкций. Инструкция по внутреннему состоянию в очередной момент работы и по буквам, обозреваемым головками (эту (n+1)-ку будем называть левой частью инструкции), определяет следующее внутреннее состояние и движение I-й,2- й,...,n- й головок (эту (n+1)-ку будем называть правой частью инструкции). Во множестве внутренних состояний выделены начальное состояние и два заключительных состояния (принимающее и отвергающее). n-ка слов принимается (отвергается), если автомат в результате ее обработки приходит в принимающее (отвергающее) заключительное состояние, и все головки при этом обозревают маркер конца.

n-ленточный 1 каа отличается от детерминированного тем, что I) программа может содержать инструкции с одинаковыми левыми частями и различными правыми частями; 2) все внутренние состояния, не являющиеся заключительными, подразделяются на экзистенциальные и универсальные. Инструкцию, левая часть которой содержит экзистенциальное (уни-

версальное) состояние, также будем называть экзистенциальной (универсальной).

Конфигурацией n-ленточного одностороннего конечного альтернирующего автомата назовем пару (q,w), где q - текущее внутреннее состояние автомата, а w - n-ка слов $w = (w_1, w_2, \ldots, w_n)$ такая, что w_i - слово, находящееся на i- й ленте от клетки, обозреваемой головкой, до маркера конца. Конфигурацию называют экзистенциальной (универсальной), если она содержит экзистенциальное (универсальное) состояние.

Путем вычисления (автомата A) называется последовательность выполняемых инструкций. Дерево (всех) путей вычисления (автомата A) - это дерево, состоящее из вершин, помеченных состояниями автомата A, которые соединены между собой дугами, помеченными инструкциями автомата A, упорядоченные таким образом, что каждый путь из корня дерева (помечен начальным состоянием автомата A) к листьям является путем вычисления. Притом для любого возможного пути вычисления такой путь в дереве существует.

Реализацией работы (автомата A со входом w) называется последовательность конфигураций, получаемых при выполнении последовательности инструкций из какого-то выполнимого на данной n-ке слов w пути вычисления.

Дерево (всех) реализаций (автомата A со входом w) - это дерево, состоящее из вершин-конфигураций, упорядоченных таким образом, что каждый путь из корня дерева (конфигурация (q_0,w), где q_0 - начальное состояние автомата A, и w - входная n-ка слов) к листьям является реализацией работы данного автомата на входной n-ке слов w. Притом для любой возможной реализации работы на этой n-ке такой путь существует.

Альтернирующий автомат A принимает n-ку слов w, если в дереве реализаций работы автомата A на данной n-ке слов w можно выделить такое конечное поддерево, что:

I) если вершина поддерева соответствует универсальной конфигурации, то вместе с ней в поддерево включаются все

ее последователи в дереве реализаций; 2) если вершина дерева соответствует экзистенциальной инструкции, то вместе с ней в поддерево включается в точности один ее последователь в дереве реализаций; 3) все листья поддерева - это конфигурации, содержащие принимающее состояние, при котором все головки обозревают маркер конца. Указанное конечное поддерево будем называть принимающим протоколом. Заметим, что выделение принимающего дерева из дерева реализаций, вообще говоря, неоднозначно.

Альтернирующий автомат называется к-ограниченным по альтернированию, если для каждого слова w принимающий протокол такой, что вдоль каждой реализации работы $c_1, c_2, \ldots, c_m$ встречается меньше, чем к переходов от экзистенциальных конфигураций к универсальным и наоборот.

Пусть A - есть альтернирующий автомат. Обозначим через T(A) язык, принимаемый автоматом A. Обозначим через $T_i(A)$ множество тех слов языка T(A), для которых существует принимающий протокол, который i-ограничен по альтернированию, т.е. в этом вдоль каждого пути вычисления встречается меньше, чем i переходов от экзистенциальных конфигураций к универсальным и наоборот.

Ниже определим понятия линейного и полулинейного подмножеств множества N^n [6]. Пусть N^n - множество неотрицательных целых чисел. Для каждого $n \geqslant 1$ положим $N^n = N \times N \times \ldots \times N$ (n раз). Введем некоторые операции. Пусть $v \in N^n$, $L_1 \subset N^n$ и $L_2 \subset N^n$. Тогда $v + L_1$ - это множество всех векторов, представимых в виде $v + u_1$, где $u_1 \in L_1$; $L_1 + L_2$ - множество векторов в виде $u_1 + u_2$, где $u_1 \in L_1$ и $u_2 \in L_2$; v^* - множество векторов в виде $k \cdot w$, где $k \in N$; $L_1 \cup L_2$ и $L_1 \cap L_2$ - это обычное объединение и пересечение множеств.

Множество $L \subset N^n$ называется линейным, если $L = w + v_1^* + v_2^* + \ldots + v_m^*$, где $w \in N^n$ и $v_i \in N^n$ $(i = 1, 2, \ldots, m)$.

Подмножество множества N^n называется полулинейным, если оно является объединением конечного числа линейных множеств.

3. ОПИСАНИЕ ПОСРЕДСТВОМ ПОЛИНЕЙНЫХ МНОЖЕСТВ ЯЗЫКОВ В УНАРНОМ АЛФАВИТЕ, ПРИНИМАЕМЫХ к-ОГРАНИЧЕННЫМИ АЛЬТЕРНИРУЮЩИМИ АВТОМАТАМИ

Будем рассматривать работу автоматов из класса n-ленточных 1 каа, работающих на языках в однобуквенном алфавите $\{1\}$.

Пусть $w = (w_1, w_2, \ldots, w_n)$ - это n-ленточное слово, $w_i \in \{1\}^*$, $i = 1, 2, \ldots, n$. Тогда на i- й ленте автомата записано слово w_i, после которого стоит один специальный маркер конца #.

Отображение Парика p(w) на данном слове $w \in (\{1\}^*)^n$ - это кортеж $(|w_1|, |w_2|, \ldots, |w_n|) \in \mathbb{N}^n$, где $|w_i|$ - это длина слова w_i на i- й ленте $(i=1,2,\ldots,n)$ [6].

Будем называть данный n-ленточный язык L полулинейным, если множество отображений Парика на словах данного языка образует полулинейное подмножество множества $\mathbb{N}^n$.

ТЕОРЕМА 3.1. Если для n-ленточного 1-стороннего конечного альтернирующего автомата, работающего в однобуквенном алфавите, существует такое к ($к \geqslant 1$), что $T(A) = T_к(A)$, т.е. все слова языка принимаются к-ограниченными по альтернированию принимающими протоколами, то T(A) - это полулинейный язык, эффективно вычислимый по A.

ДОКАЗАТЕЛЬСТВО. Выделим из множества S всех состояний автомата A следующую последовательность подмножеств: $S_1, S_2, \ldots, S_k$. Для каждого $i=1,2,\ldots,k$ множество S_i состоит либо только из экзистенциальных, либо только из универсальных состояний, при этом S_{i+1} - это множество всех тех состояний из S, которые достижимы из состояний множества S_i по всем таким путям вычислений, которые содержат в точности одну перемену кванторов (из универсальных состояний на экзистенциальные или наоборот). Без ограничений к общности можем предположить, что начальное состояние q_0 - экзистенциальное. Тогда множество S_1 состоит из всех экзистенциальных состояний, которые достижимы из состояния q_0 по всем путям вычисления без перемены кванторов.

Пусть $q \in S$. Обозначим через $L(q)$ множество всех слов, принимаемых автоматом A, если считать состояние q начальным. В этих обозначениях выполняется $L(q_0) = T(A)$.

Пусть $q \in S_i$. Обозначим через $L_i(q)$ $(i = 1,2,\ldots,k)$ множество всех слов, принимаемых автоматом A, если считать начальным состояние q, для которых принимающие протоколы $(k-i)$-ограничены по альтернированию (т.е. в каждой реализации принимающего протокола от корня до листьев имеется не более $(k-i)$ перемен кванторов). Для $q \in S \setminus S_i$ определяем, что $L_i(q) = \emptyset$ (пустое множество).

Для $q \in S_i$ через $L_0(q)$ обозначим множество слов, принимаемых автоматом A, если q – начальное состояние, и принимающие протоколы не содержат перемены кванторов.

Для $d \in N$ обозначим через Z_d следующее множество n-ленточных слов: $\{w \mid$ существует $i \in \{1,2,\ldots,n\}: |w_i| \leq d, w = (w_1, w_2, \ldots, w_n)\}$.

Допустим, что для всех состояний $q \in S$ известны множества $L(q) \cap Z_d$ и это полулинейные языки. Из того, что $T(A) = T_k(A)$, следует, что $L(q_0) = L_k(q_0) \cup L_{k-1}(q_0) \cup \ldots \cup L_1(q_0)$.

Покажем, что для любого $q \in S_i$, $i \in \{1,2,\ldots,k\}$, множество слов $L_i(q)$ является полулинейным языком, из чего будет следовать полулинейность $L(q_0)$.

Доказательство этого мы приведем индукцией по i $(i = k, k-1, \ldots, 1)$.

Для $i = 2,\ldots,k$ через B_i обозначим все те состояния из множества S_i, которые достижимы из состояний множества S_{i-1} с помощью в точности одной инструкции. Множество $B_1 = \{q_0\}$.

Индуктивное предположение. Допустим, что для всех состояний $q \in S_{i+1}$ установлены множества слов $L_{i+1}(q)$ и это полулинейные языки.

Индуктивный переход. Рассмотрим более простой случай, когда множество S_i содержит только экзистенциальные состояния.

Пусть $q \in S_i$. Все принимающие протоколы, начинающиеся в состоянии q, можно разделить на два типа. Первые –

это те реализации вычисления, которые не содержат перемен кванторов, т.е. доходят до принимающего заключительного состояния, выходя из подмножества S_i. Все слова, принимаемые по этим реализациям, - это множество $L_0(q)$. Вторые - это те, принимающие протоколы, которые содержат хотя бы одну перемену кванторов.

Пусть q' - состояние в принимающем протоколе после первой перемены кванторов. Тогда $q' \in B_{i+1}$. Если принимаемый протокол с корнем (q,w) в начальной части с экзистенциальными конфигурациями после первой перемены кванторов содержит конфигурацию с состоянием q', то n-ку слов w можно описать следующим образом: $p(\Delta L_i(q,q'))+p(L_{i+1}(q'))$, где $\Delta L_i(q,q')$ - это множество n-ленточных подслов, соответствующих всем путям вычисления из состояния q до состояния q', таких, что все промежуточные состояния принадлежат множеству S_i. Согласно леммам 3.2, 3.I из [4] это множество слов полулинейно и эффективно построимо. Поэтому полулинейно и эффективно построимо множество $p(L_i(q))=\bigcup_{q' \in B_{i+1}} [p(\Delta L_i(q,q'))+p(L_{i+1}(q'))] \cup p(L_0(q))$ эффективно построимое полулинейное подмножество множества N^n.

Рассмотрим более сложный случай, когда S_i состоит только из универсальных состояний. Покажем полулинейность и эффективную построимость множества $L_i(q)$, $q \in S_i$.

По определению, за каждым универсальным состоянием в принимающий протокол входят все из возможных продолжений из дерева всех реализаций.

Будем говорить, что реализация при данной конфигурации (q',w') достигла зоны Z_d, если $w' \in Z_d$.

Все реализации из принимающего протокола можно разделить на две группы: одна группа - это те реализации, которые достигают зону Z_d без перемены кванторов, и вторая группа - это те реализации, при которых квантор состояния меняется, не доходя до зоны Z_d.

Приведем два утверждения:

(I) все реализации первой группы должны быть принимающими. Во-первых, если существует реализация, которая при-

водит автомат из конфигурации (q,w) в конфигурацию (q',w'), где $q,q' \in S_i$, $w' \in Z_d$, и промежуточные конфигурации также содержат состояния из S_i, то $w' \notin z_d^i(q')$ влечет $w \notin L_i(q)$, где $z_d^i(q)$ - множество всех таких слов w из Z_d, которые принимаются принимающими протоколами с корнем (q,w), и в которых вдоль каждой реализации имеется не более (k-i) перемен кванторов. Во-вторых, каждая реализация из дерева всех реализаций со входом w должна либо дойти до зоны Z_d, не меняя квантор, либо переменить квантор, не доходя до зоны Z_d, т.е. слово w не принимается, если дерево всех реализаций с корнем (q,w) содержит обрывающуюся реализацию, содержащую только универсальные состояния из S_i.

(2) все реализации из второй группы должны быть принимающими. Если существует реализация, которая переводит автомат из конфигурации (q,w) в конфигурацию (q',w'), где $q' \in B_{i+1}$, и $w,w' \notin Z_d$, то из $w' \notin L_{i+1}(q')$ следует $w \notin L_i(q)$.

Рассмотрим теперь множество слов, отображение Парика которых выражается следующим образом:

$$(*)\quad \mathbb{N}^n \setminus \bigcup_{q' \in S_i} [p(\Delta L(q,q')) + p(\overline{Z_d^{i+1}(q')})] \setminus \quad (i)$$

$$\setminus \bigcup_{q' \in B_{i+1}} [p(\Delta L(q,q')) + p(\overline{L_{i+1}(q')})] \setminus \quad (ii)$$

$$\setminus \bigcup_{q' \in S_i \cap B} [\mathbb{N}^n + p(\Delta L(q,q'))] \setminus \quad (iii)$$

$$\setminus p(Z_d) \cup p(Z_d^i(q)), \quad (iiii)$$

где B - все тупиковые, незаключительные состояния.

Покажем, что при условии, что для каждого $q \in S$ множество $p(z_d^i(q))$ известно и является полулинейным, то выражение (*) - это множество $p(L_i(q))$.

Для слов из Z_d - это по определению $Z_d^i(q)$ (см.(iiii)).

Пусть слово $w \notin Z_d$. Рассмотрим то поддерево из дерева всех реализаций с корнем (q,w), состояния которых не выходят из S_i. Каждая реализация этого поддерева либо доходит до какой-то конфигурации (q',w'), где $q' \in S_i$, $w' \in Z_d$, либо переходит в какую-то конфигурацию (q',w'), где $q' \in B_{i+1}$ ((iii) гарантирует, что ни одна реализация поддерева не обрывается в тупике). Если лист рассматриваемого поддерева - конфигурация (q',w'), где $q' \in S_i$ и $w' \in Z_d$, то (i) гарантирует $w' \in Z_d^i(q')$, и эта ветвь - принимающая.

Если лист - конфигурация (q',w'), где $q' \in B_{i+1}$, то (ii) гарантирует $w' \in L_{i+1}(q')$, и эта ветвь также принимающая.

Покажем, что для $q \in S_i$ множество $p(Z_d^i(q))$ полулинейное, эффективно построимое подмножество множества $p(Z_d)$.

Пусть имеется n-ленточный 1 каа A. Этот автомат мы можем разделить на два подавтомата A_1 и A_2. В A_1 входят все те инструкции, которые предполагают, что ни одна из n головок не видит маркер конца, а в A_2 - все остальные инструкции. В свою очередь, в A_1 можно выделить подавтоматы A_{11}, $A_{12},\ldots,A_{1n}$, где A_{1j} $(j = 1,2,\ldots,n)$ - это подавтомат, инструкции которого предполагают, что головка i-ой ленты стоит на месте и обозревает символ алфавита. Аналогично в A_2 можно выделить подавтоматы $A_{21}, A_{22},\ldots, A_{2n}$, где A_{2j} $(j = 1,2,\ldots,n)$ - это подавтомат, инструкции которого предполагают, что головка j-ой ленты стоит на месте и обозревает маркер конца слова на этой ленте. Так как автомат A работает на слове $w \in Z_1$, то имеется $j \in \{1,2,\ldots,n\}$ такое, что слово на j-ой ленте либо пустое, либо состоит из одного символа. Тогда в любой реализации после работы подавтомата A_{1j} имеется в точности одна инструкция, передвигающая головку j-ой ленты вправо, после чего начинает работать подавтомат A_{2j}.

Таким образом, задачу нахождения $Z_1^i(q)$ для n-ленточно-

го автомата можно свести к паре по сути (n-1)-ленточных подавтоматов A_{1j} и A_{2j}, при работе которых передвижение по j-ой ленте не происходит.

Если в подавтомате из всех n головок передвигается только головка одной ленты, а остальные обозревают маркер конца, то из [1] легко видно, что для каждого состояния q этого подавтомата и каждого $i = k, k-1, \ldots, 2, 1$ язык $L_i(q)$ регулярный и эффективно построенный. Соответственно, множество $p(L_i(q))$ полулинейно и эффективно построимое подмножество N^n.

Используя для подавтоматов с некоторыми стоящими головками выражение (*) для $L_i(q)$ и индукцию по числу лент, по которым головкам разрешается передвижение, можно построить множество $Z_1^i(q)$. Подставив в выражение (*) $d = 1$, получаем, что множество $L_1(q)$ выражается через множества $Z_1^{i+1}(q')$, $L(q,q')$ и множества $L_{i+1}(q')$, которые по индуктивному предположению известны. Индуктивный переход завершен.

4. СРАВНЕНИЕ n-ЛЕНТОЧНЫХ 1каа С ОГРАНИЧЕНИЕМ ПО АЛЬТЕРНИРОВАНИЮ И n-ЛЕНТОЧНЫХ 1кнα, РАБОТАЮЩИХ В ОДНОБУКВЕННОМ АЛФАВИТЕ

ТЕОРЕМА 4.1. Класс языков в однобуквенном алфавите, принимаемых n-ленточными 1-сторонними конечными альтернирующими автоматами с константным ограничением по альтернированию, совпадает с классом языков, принимаемых n-ленточными 1-сторонними конечными недетерминированными автоматами.

ДОКАЗАТЕЛЬСТВО. Теорема 3.1 утверждает, что язык, принимаемый n-ленточным 1 каа, к-ограниченным по альтернированию, - это полулинейный язык. Из лемм 3.2 и 3.1 из [4] следует, что язык, принимаемый n-ленточным1 кна, также является полулинейным. Так как рассматриваются языки в однобуквенном алфавите, классы этих языков совпадают, что и требовалось доказать.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. **Chandra A.K., Kozen D.C. and Stockmeyer L.J. Alternation//Journal of the ACM, 1981. V.28. P.114-133.**
2. **Freivalds R. Projections of languages Recognizable by Probabilistic and Alternating Finite Multitape Automata //Information Processing Letters, 1981. V.13. P.195-198.**
3. Гейдманис Д.Г. Об одной мере сложности для альтернирующих конечных многоленточных автоматов//Сложностные проблемы математической логики. Калинин, 1985. С.25 - 27.
4. Гейдманис Д.Г. О возможностях альтернирующих и параллельно работающих недетерминирующих конечных многоленточных автоматов//Латвийский математический ежегодник. Рига, 1988. Вып.31. С.115-127.
5. **Geidmanis D. On the Capabilities of Alternating and Nondeterministic Multitape Automata//Proc.Fund. of Comp.Theory, Lect.Notes in Comp.Sci. Springer, 1988. V.278. P.150-154.**
6. Гинзбург С. Математическая теория контекстно-свободных языков. М.: Мир, 1970.
7. **K.N.King. Alternating Multihead Automata// Lect. Notes in Comp.Sci. Springer, 1981. V.115. P.506-520.**
8. Мучник А.А. Добавление переводчика к статьям "Об альтернировании, I, II"//Кибернетический сборник. М.: Мир, 1983. Вып.20.С.141-158.
9. Рабин М.О., Скотт Д. Конечные автоматы и задачи их разрешения// Кибернетический сборник. М.: Мир, 1962. Вып. 4. С.58-91.

О ЧИСЛЕ ЛИСТЬЕВ ПРИНИМАЮЩЕГО ДЕРЕВА ДЛЯ ОДНОСТОРОННИХ АЛЬТЕРНИРУЮЩИХ МАГАЗИННЫХ АВТОМАТОВ

Я.Я.Канеп
ВЦ при ЛГУ им. П.Стучки

Понятие альтернирующих автоматов было введено в [I] как обобщение недетерминированных автоматов.

Подробное определение детерминированных односторонних магазинных автоматов можно найти в [2]. Они имеют входную ленту, на которой головка может стоять на месте или двигаться слева направо, и один магазин. Работа автомата задается программой, состоящей из команд. Команды по внутреннему состоянию в очередной момент работы и по буквам, обозреваемым головками на входной ленте и на магазине (эту тройку будем называть левой частью команды), определяет следующее внутреннее состояние, движения головок и, возможно, букву, записываемую в магазин (эту четверку будем называть правой частью команды). На входной ленте вправо от входного слова имеется правый концевой маркер, а в магазине - выделенный маркер дна. Во множестве внутренних состояний выделены начальное состояние и два заключительных состояния (принимающее и отвергающее). Входное слово принимается (отвергается), если в результате его обработки автомат приходит в принимающее (отвергающее) заключительное состояние.

Иногда нам будет удобно считать, что в начале работы автомата в магазине вправо от маркера дна записано некоторое слово w. Тогда, если v - входное слово, то в зависимости от исхода обработки, будем говорить, что автомат принимает или отвергает пару (v, w). В частности, слово v принимается, если принимается пара (v, ε), где ε - пустое слово.

Альтернирующий односторонний магазинный автомат отличается от детерминированного тем, что: I) программа может содержать команды с одинаковыми левыми частями и различными правыми частями; 2) все внутренние состояния, не являющиеся заключительными, подразделяются на экзистенциальные и универсиальные. Команду, левая часть которой содержит экзистенциальное (универсальное) состояние, также будем называть экзистенциальной (универсальной).

Реализацией работы автомата на входном слове (паре слов) называется последовательность выполняемых команд. Дерево реализаций работы автомата состоит из команд, упорядоченных таким образом, что каждый путь из корня дерева к листьям является реализацией автомата на данном слове (паре слов), притом такой путь существует для любой возможной реализации работы на этом слове (паре слов).

Альтернирующий автомат принимает данное слово (пару слов), если в дереве реализаций можно выделить такое конечное поддерево, что:

I) если вершина поддерева соответствует универсальной команде, то вместе с ней в поддерево включаются все ее последователи в дереве реализаций;

2) если вершина поддерева соответствует экзистенциальной команде, то вместе с ней в поддерево включается хотя бы один ее последователь;

3) все листья поддерева соответствуют принимающим состояниям. Указанное поддерево будем называть принимающим деревом работы автомата на данном слове (паре слов). Отметим, что в общем случае дерево реализаций может содержать более одного такого поддерева.

Будем говорить, что альтернирующий автомат имеет тип $\forall$, если все его состояния универсальны. Автомат имеет тип $\exists\forall$, если при любой реализации работы на любом слове все экзистенциальные команды предшествуют всем универсальным командам.

Будем говорить, что автомат принимает слово (пару слов) с числом листьев k, если для этого слова(пары слов)

существует принимающее дерево работы данного автомата с числом листьев k .

Для языка слов L будем считать, что он распознается автоматом с числом листьев $f(l)$, если:

1) каждое $w \in L$ длины l принимается с числом листьев не более $f(l)$;

2) ни для какого $w \tilde{\in} L$ не существует принимающего дерева.

Описанная мера сложности в некотором смысле показывает число универсальных команд, необходимых для распознавания. Отметим, что недетерминированными автоматами языки принимаются с числом листьев 1.

Из теоремы Парика /2/ вытекает, что любой язык в однобуквенном алфавите, принимаемый недетерминированным и, тем более детерминированным, односторонним магазинным автоматом, является регулярным. Для альтернирующих автоматов это не так. В настоящей статье в виде двух теорем приводится результат о минимальном числе листьев принимающего дерева, необходимом для распознавания нерегулярных языков в однобуквенном алфавите такими автоматами. При этом оказывается, что нижнюю границу - двойной логарифм от длины слова можно достигнуть автоматами типа $\exists\forall$.

В дальнейшем все численные переменные являются целыми.

Пусть даны $k \geq 2$, $\alpha > 0$, $q > 0$ и $(\alpha_k \alpha_{k-1} \ldots \alpha_0 - k)$-ричное разложение α (т.е. $\alpha = \sum_{i=0}^{k} \alpha_i k^i, \forall i (\alpha_i \in [0, k-1]), \alpha_k > 0$). Через $W_k(\alpha, q)$ обозначим слово $\alpha^q \alpha_0 \alpha^{2q} \alpha_1 \ldots \alpha^{(k+1)q} \alpha_k$ в алфавите $\{0, 1, \ldots, k-1\} \cup \{\alpha\}$.

ЛЕММА. Для любого $k \geq 2$ существует альтернирующий односторонний магазинный автомат типа $\forall$ $\mathcal{A}_k$ со свойствами:

1) входным алфавитом $\mathcal{A}_k$ является $\{0\}$;

2) алфавит магазина $\mathcal{A}_k$ - множество $\{\alpha, \beta\} \cup \{0, 1, \ldots, k-1\}$; β-маркер дна; 3) $\mathcal{A}_k$ принимает пару $(0^l, v)$ тогда и только тогда, если $\exists \alpha > 0$ $\exists q > 0$ $(l = \alpha q \;\&\; v = W_k(\alpha, q))$;

4) каждая пара $(\alpha q, W_k(\alpha, q))$ принимается с числом листьев $[\log_k \alpha] + 2$.

ДОКАЗАТЕЛЬСТВО. Пара $(0^l, v)$ должна приниматься тогда и только тогда, если выполняются условия:

1) слово v представимо в форме $\alpha^{c_0}\alpha^{c_1}\alpha_1 \ldots \alpha^{c_k}\alpha_k$ для некоторых $k \geq 0$, $\alpha_k > 0$, $\forall i\,(\alpha_i \in [0, h-1] \,\&\, c_i > 0)$;

2) $\forall i\,(c_i = c_0 h^i)$; 3) $l = \sum_{i=0}^{k} \alpha_i c_i$.

Опишем работу автомата $\mathcal{A}_h$. В любой реализации автомат головку магазина передвигает справа налево и проверяет первое условие. Если это условие не выполняется, то пара $(0^l, v)$ отвергается.

Пусть первое условие выполняется. Заметим, что тогда остальные два условия эквивалентны системе равенств

$$\begin{cases} (1) \quad l = \sum_{i=0}^{k} \alpha_i c_i \\ (2) \quad \forall j \in [0, k-1]\,\Big(l = \sum_{i=0}^{j-1} \alpha_i c_i + (\alpha_j + h) c_j + \sum_{i=j+1}^{k-1} (\alpha_i + h - 1) c_i + \\ \qquad + (\alpha_k - 1) c_k\Big) \end{cases}$$

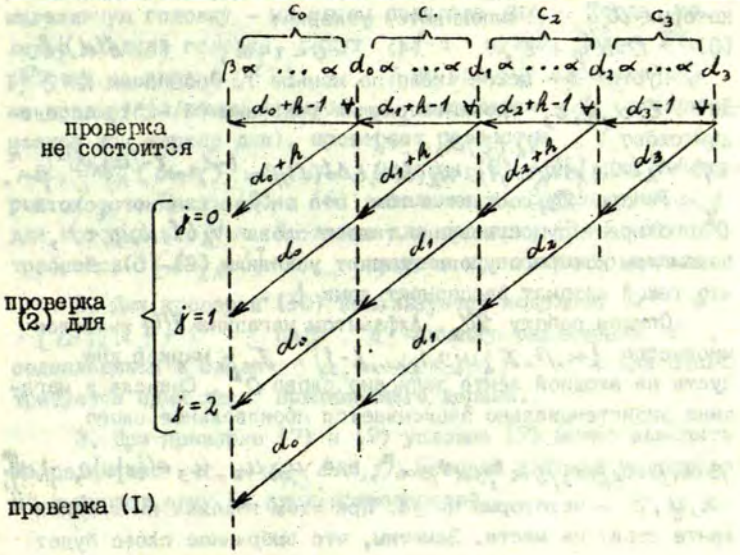
При $k = 0$ в системе присутствует лишь равенство $l = \alpha_0 c_0$.

Будем говорить, что во время движения головки магазина через подслово α^{c_i} $(i \in [0, k])$ автомат работает с начальной, первой, второй или концевой скоростью, если за это время входная головка передвигается соответственно на $(\alpha_i - 1)c_i$, $(\alpha_i + h - 1)c_i$, $(\alpha_i + h)\,c_i$ или $\alpha_i c_i$ нулей вправо. Автомат в ходе работы универсально выбирает проверяемое равенство системы. Это достигается следующим образом. В начале автомат выбирает, будет ли он проверять равенство (1) или (2) для некоторого j. В первом случае входная головка все время передвигается с концевой скоростью. Рассмотрим второй вариант. Тогда во время прохода через слово α^{c_k} применяется начальная скорость. Если автомат, работая с начальной или первой скоростью встречает маркер дна β, то пара $(0^l, v)$ принимается, так как эта реализация не соответствует ни одному равенству системы (I), (2). Если автомат, работая с начальной или первой скоростью, на магазине достигает символ α_j $(j \in [0, k-1])$, то он для дальнейшей работы

универсально выбирает первую или вторую скорость. Вторая скорость применяется либо до встречи маркера дна β (если $j=0$), либо до встречи следующей буквы d_{j-1} (если $j>1$). После достижения d_{j-1} вторая скорость заменяется на концевую, которая применяется до встречи маркера дна β. Рассмотренная реализация проверяет справедливость равенства (2) при соответствующем j.

При проверке каждого равенства пара $(0^k, v)$ принимается, если головка входной ленты достигает концевой маркер одновременно с достижением маркера дна головкой магазина.

Схема движения головки магазина автомата $\mathcal{A}_k$ при $k=3$ показана на рисунке. К стрелкам приписаны отношения скоростей движения головок входной ленты и магазина на соответствующих этапах. Символом "$\forall$" помечены точки, соответствующие моментам универсального выбора.



Заметим, что число листьев принимающего дерева на любой паре $(\alpha q, W_k(\alpha,q))$, где $W_k(\alpha,q)=\alpha^q\alpha_0\alpha^{qk}\alpha_1\ldots\alpha^{qk^s}\alpha_s$, на один больше числа равенств системы (1),(2), т.е. равно $\kappa+2=[\log_2(\sum_{i=0}^{s}\alpha_i k^i)]+2=[\log_2\alpha]+2$.

Лемма доказана.

Далее под $\beta|\alpha$ понимается свойство "α делится на β".

ТЕОРЕМА 1. Существует нерегулярный язык L в однобуквенном алфавите и последовательность альтернирующих односторонних магазинных автоматов типа $\exists\forall$ $\mathcal{M}_1,\mathcal{M}_2,\ldots$ такая, что при всех s $\mathcal{M}_s$ распознает L с числом листьев $\log_2(\log_2\ell)/s$ при $\ell>\ell_s$ (ℓ_s - константа, зависящая от s).

ДОКАЗАТЕЛЬСТВО. Определим

$$L=\{0^\ell \mid \exists\alpha>0(\gamma(\alpha|\ell)\&(2^{[\log_2\alpha]+1}))\}.$$

В дальнейшем для $\alpha>0$ определим функцию $\beta(\alpha)=2^{[\log_2\alpha]+1}$. Заметим, что $0^\ell\in L$ тогда и только тогда, когда для некоторых α,q,τ выполняются условия:

(3) $\ell=\alpha q+\tau$, (4) $0<\tau<\alpha$, (5) $\beta(\alpha)|\ell$.

Пусть s - целое число не меньше 1. Обозначим $k=2^{4s}$. Для ℓ,α,q,τ, удовлетворяющих условиям (3)-(5), определим слово $V_k(\ell,\alpha,q,\tau)=$
$=\beta W_k(\alpha,q)\beta W_k(\alpha,1)\beta W_k(\beta(\alpha),\ell/\beta(\alpha))\beta\alpha^{\ell-\alpha}\beta\alpha^{\ell-[\log_k\alpha]-1}\beta\alpha^{\tau}$.

Работа $\mathcal{M}_s$ состоит в том, что он для входного слова 0^ℓ проверяет существование такого слова $V_k(\ell,\alpha,q,\tau)$, параметры которого удовлетворяют условиям (3)-(5). Ясно, что такой автомат распознает язык L.

Опишем работу $\mathcal{M}_s$. Алфавитом магазина $\mathcal{M}_s$ является множество $\{\alpha,\beta,\gamma\}\cup\{0,1,\ldots,k-1\}$, γ - маркер дна. Пусть на входной ленте записано слово 0^ℓ. Сначала в магазине экзистенциально записывается произвольное слово

$\beta u_1\beta u_2\beta u_3\beta\alpha^x\beta\alpha^y\beta\alpha^z$, где $u_1,u_2,u_3\in(\{\alpha\}\cup\{0,\ldots,k-1\})^*$;
x,y,z - некоторые числа. При этом головка на входной ленте стоит на месте. Заметим, что выбранное слово будет

равно $V_k(l,\alpha,q,\tau)$ для l,α,q,τ, удовлетворяющих условиям (3)-(5), тогда и только тогда, когда дополнительно к (3)-(5) будут в силе утверждения:

(6) $u_1 = W_k(\alpha,q)$,

(7) $u_2 = W_k(\alpha,1)$,

(8) $u_3 = W_k(b(\alpha),l;b(\alpha))$,

(9) $x = l-\alpha$,

(10) $y = l-[\log_k\alpha]-1$.

Проверка условий (3)-(10) производится универсально. Действия $\mathcal{M}_5$ изложим в пяти пунктах. При рассмотрении каждого следующего пункта будем считать, что условия, проверка которых описана в предыдущих пунктах, выполняются.

1. Для проверки (3) и (6) автомат, используя слово α^τ, передвигает входную головку на τ нулей вправо, а магазинную головку - на конец подслова βu_1. Тогда не левее входной головки будут $l-\tau$ нулей. Далее $\mathcal{M}_5$, работая на паре $(0^{l-\tau}, u_1)$ по образцу автомата $\mathcal{A}_k$ из предыдущей леммы (символ β автоматом $\mathcal{A}_k$ воспринимается как маркер дна), проверяет равенства $l-\tau=\alpha q$, $u_1 = W_k(\alpha,q)$, т.е. условия (3) и (6) для некоторых α,q. Пусть (3) и (6) выполняются. Зафиксируем α,q для которых это так. По лемме для проверки (3) и (6) потребуется $[\log_k\alpha]+2$ листьев принимающего дерева.

2. Для проверки (10) используется подслово α^y и $[\log_k\alpha]+1$ цифр k-ричного разложения α, содержащихся в слове $u_1 = W_k(\alpha,q)$. Для этого требуется один лист принимающего дерева.

3. При проверке (7) и (9) условие (7) можно заменить на равенство $u_2 = W_k(l-x,1)$. Сначала автомат универсально выбирает одну из двух возможностей.

В первом случае проверяется равенство $u_z = W_h(l-x,1)$. Для этого входная головка передвигается на x нулей вправо (используется α^x), и головка магазина переводится на конец подслова βu_z. Далее автомат, работая на паре $(0^{l-x}, u_z)$ по образцу $\mathcal{A}_h$, проверяет условие $u_z = W_h((l-x)/i, i)$ для некоторого i. Для этого потребуется $[\log_h((l-x)/i)]+2$ реализаций. В тех же реализациях дополнительно детерминированно проверяется равенство $i = I$.

Пусть в первом случае все реализации завершатся успешно, т.е. $u_z = W_h(l-x,1)$. Во втором случае проверяется (9), т.е. $l-x = \alpha$. Для этого достаточно проверить два свойства: 1) число цифр h-ричного разложения $l-x$ равно $[\log_h \alpha]+1 = l-y$,
2) для всех i $(0 < i \le [\log_h \alpha]+1)$ i - я цифра h-ричного разложения $l-x$ равна i-той цифре разложения α. Для проверки этих свойств используются слова

$$u_1 \beta u_z = W_h(\alpha, y) \beta W_h(l-x,1), \quad \alpha^y = \alpha^{l-[\log_h \alpha]-1}.$$

Сначала, используя α^y, входная головка передвигается на позицию, не левее которой находятся $[\log_h \alpha]+1$ нулей. Потом в одной из реализаций проверяется, совпадает ли число этих нулей с числом цифр h-ричного разложения $l-x$, содержащейся в слове u_z. Пусть это верно. В других реализациях головка магазина универсально переводится на произвольную h-ричную цифру разложения $l-x$. Входная головка при этом не двигается. Выбранная цифра запоминается. Потом головка магазина передвигается справа налево, причем входная головка переводится на I нуль вправо в моменты встреч какой-либо h-ричной цифры. Это продолжается до тех пор, пока на входной ленте не достигается концевой маркер. В конце проверяется, совпадает ли текущая h-ричная цифра с выбранной в начале. Всего для проверки (9) потребуется $[\log_h \alpha]+2$ реализаций.

Значит, в пункте 3 используются всего

$$[\log_h(l-x)]+2+[\log_h \alpha]+2 = 2[\log_h \alpha]+4$$

листьев принимающего дерева.

4. Из (9) следует, что для проверки (4) достаточно проверить неравенства $z>0$ и $z+x<\ell$. Проверка происходит очевидным образом (используются слова α^x, α^z). Для (4) используется один лист принимающего дерева.

5. Остались неописанными проверки условий (5) и (8). Заметим, что (5) следует из (8). Для проверки (8) M_5 работает на паре $(0^\ell, u_3)$ по образцу A_k из леммы. Таким образом проверяется равенство $u_3 = W_k(i, \ell/i)$ для некоторого i. Пусть оно верно, тогда для его проверки используются $[\log_k i]+2$ реализаций. Кроме того в одной реализации проверяется равенство $i = b(\alpha)$. Заметим, что $i = b(\alpha)$ верно, если выполняется одно из следующих двух утверждений:

I) разложение α имеет вид $\alpha_\kappa \alpha_{\kappa-1} \dots \alpha_0$, а разложение i – вид $10\underbrace{\dots 0}_{\kappa+1}$, где $\kappa = [\log_k \alpha]$, $\alpha_\kappa \geq k/2$;

2) разложение α имеет вид $\alpha_\kappa \alpha_{\kappa-1} \dots \alpha_0$, а разложение i – вид $b(\alpha_\kappa)\underbrace{0\dots 0}_{\kappa}$, где $\kappa = [\log_k \alpha]$, $\alpha_\kappa < k/2$.

Очевидно, что дизъюнкция этих условий может быть проверена детерминированно одной реализацией. Для этого используются слово $\alpha^y = \alpha^{\ell - [\log_k \alpha] - 1}$ и разложения α и i, содержащийся в словах $u_1 = W_k(\alpha, q)$, $u_3 = W_k(i, \ell/i)$. Всего для пятого пункта используются $[\log_k (b(\alpha))]+3 \leq [\log_k \alpha]+4$ листьев.

Подводя итоги, всего принимающее дерево, проверяющего условия (3)-(10), имеет не более $4[\log_k \alpha]+12$ листьев.

Обозначим через α_{min} наименьшее из значений α, для которых $7(\alpha/\ell)$. Очевидно, если $0^\ell \in L$, то для α_{min} верно и $b(\alpha_{min}) | \ell$, так как $(\alpha_{min} \leq \alpha) \supset (b(\alpha_{min}) | b(\alpha))$. Следовательно, слово 0^ℓ принимается с числом листьев не более $4[\log_k \alpha_{min}]+12$.

Для оценки α_{min} применим метод Фрейвалда из /3/. Пусть каноническое разложение ℓ на простые множители имеет вид $p_1^{m_1} p_2^{m_2} \dots p_t^{m_t}$. Тогда α_{min} не больше наименьшего про-

стого числа, отличного от $p_1, p_2, \dots, p_t$, это означает, что d_{min} не превышает простого числа, номер которого в естественной последовательности равен $t+1$:

$$l = p_1^{m_1} p_2^{m_2} \dots p_t^{m_t} \geq t!$$

$$t \leq const \log_2 l / \log_2 \log_2 l$$

По теореме Чебышева о частоте простых чисел $d_{min} \leq const \log_2 l$. Значит, слово 0^l принимается с числом листьев не более

$$4[\log_2 (const \log_2 l)] + 12 \leq 5 \log_2 \log_2 l / \log_2 k = \\ = \log_2 \log_2 l / 5$$

при достаточно больших l.

Заметим, что автомат $\mathcal{M}_5$ имеет тип $\exists\forall$, так как при любой реализации его работы после экзистенциальной записи в магазине слова $\beta u_1 \beta u_2 \beta u_3 \beta \alpha^x \beta \alpha^y \beta \alpha^z$ экзистенциальный выбор больше не применяется. Теорема доказана.

ТЕОРЕМА 2. Если альтернирующий односторонний магазинный автомат распознает язык L в однобуквенном алфавите с числом листьев $\sigma(\log_2 \log_2 l)$ (l - длина входного слова), то язык L регулярен.

Доказательство здесь не приводится.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Chandra A.K., Kozen D.C., Stockmeyer L.J. Alternation // J.Ass.Comput.Mach. 1981. Vol.28. N 1. P. 114-133.

2. Гладкий А.В. Формальные грамматики и языки. М., 1973.

3. Фрейвалд Р.В. О времени работы детерминированных и недетерминированных машин Тьюринга // Латвийский математический ежегодник. Рига, 1979, Вып. 23, С.158-165.

РАСПОЗНАВАНИЕ ИЗОМОРФИЗМА ГРАФОВ ИНТЕГРАЛЬНЫХ СХЕМ

П.Б.Ручевскис

ВЦ при ЛГУ им.П.Стучки

В последнее десятилетие в вычислительной науке наблюдается значительный прогресс в сторону более глубокого понимания характера сложности проблемы распознавания изоморфизма графов. Построены полиномиальные алгоритмы для ряда классов графов с ограниченными параметрами [3,13], и получены полиномиальные оценки сверху времени, требуемого для распознавания изоморфизма почти всех графов [7] .

По утверждению [3] в настоящее время проблема изоморфизма заключается в оценке сложности самой проблемы, т.е. лучших алгоритмов изоморфизма. К сожалению, как признают и сами авторы этих алгоритмов [7] , полученные результаты еще далеки от удовлетворения нужд практических приложений. Препятствиями оказываются или слишком высокая степень полиномов, оценивающих требуемое время работы, или явное исключение из рассмотрения как раз графов, возникающих в приложениях. Поэтому для практики более приемлемыми оказались алгоритмы, основанные на использовании различных эвристик:

Самой распространенной из них является процедура стабилизации раскраски вершин [1,8,10] , которая, к тому же признана важным теоретическим инструментом. Комбинирование механизма стабилизации с различными стратегиями перебора является типичной чертой всех практически используемых алгоритмов распознавания изоморфизма графов.

Практические вычисления на ЭВМ дополнительно требуют разработки эффективных структур данных для программ, реализующих выбранные алгоритмы.

Именно последнее и составляет основу настоящей рабо-

ты, посвященной рассмотрению алгоритма, сконструированного в духе [12] с учетом соображений [3] , и далее развивающего традиции работы [4] .

I. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Будем придерживаться стандартной графо-теоретической терминологии [6] и далее рассмотрим только неориентированные раскрашенные графы $G=(V,E,P)$, где

V – множество вершин графа ;

E – множество ребер графа ;

P – раскраска (разбиение) вершин графа .

Обозначим через $adj(v)$ – множество вершин смежных с вершиной v .

Раскраской P множество вершин V разбивается на классы цветов $V_i=\{v\in V: P(v)=i\}$ $i=1,\dots,k$, где k число цветов.

Определение 1 .

Цвет i назовем однородным относительно цвета j, если $\forall v\in V_i$ $(|adj(v)\cap V_j|=r$ (константа, не зависящая от v)).

Определение 2 .

Раскраску назовем стабильной, если для любых i , j цвет j однороден относительно цвета i .

Обозначим через P_{ji} подразбиение цвета j относительно цвета i : $P_{ji}=\{V_{j1},\dots,V_{jr}\}$, где $V_{jq}=\{v\in V_j: |adj(v)\cap V_i|=q$ и $|V_{jq}|>0\}$, $r\in\{1,\dots,|V_i|\}$.

Определение 3 .

Для двух графов $G1$ и $G2$ будем говорить, что их подразбиения $P(G1)_{ji}$ и $P(G2)_{ji}$ согласованны, если $\forall t$ $|V(G1)_{jt}|=|V(G2)_{jt}|$, $V(G1)_{jt}\in P(G1)_{ji}$ и $V(G2)_{jt}\in P(G2)_{ji}$.

Определение 4.

Раскрашенные графы $G1$ и $G2$ называются изоморфными тогда и только тогда, когда существует такое взаимно однозначное соответствие между их вершинами f , что $\forall(x,y)\in E(G1)$ $((f[x],f[y])\in E(G2))$ и $\forall x\in V(G1)$ $(P(G1)[x]=P(G2)[f[x]])$.

Лемма I.

Раскрашенные графы $G1$ и $G2$ изоморфны тогда и только тогда, когда эти графы изоморфны при согласованной стабильной раскраске.

Доказывается непосредственно.

2. АЛГОРИТМЫ РАСПОЗНАВАНИЯ ИЗОМОРФИЗМА

Мы остановились на следующей версии алгоритма распознавания изоморфизма (в основу взята лемма I и объединены важнейшие черты [3,4,I2]).

```
procedure  Изоморфны (G01 , G02 returns (Boolean);
Input  : раскрашенные графы G01  и G02 .
Output  : если графы изоморфны тогда значение  true,
          иначе  false.
   begin
    G1: = G01;
    G2: = G02;
    if Разбиение (G1  , G2) then return (false);
    if  V(G1)=0 then return (true);
x:=  некоторая вершина графа  G1;
    i:= P(G1) [x] ;
    P(G1) [x] := неиспользованный цвет;
    for y in V(G2)i do
    begin
       P(G2) [y] := P(G1) [x] ;
       if Изоморфны  (G1, G2) then return (true);
       P(G2) [y] :=i;
    end;
    return (false);
   end.
   procedure  Разбиение       (G1,G2) returns (Boolean);
```

```
Input:  раскрашенные графы G01 и G02.
Output: если стабильная раскраска согласована
        тогда значение true , иначе false.
 begin
  if  ¬ Согласована стабильная раскраска
     then return (false);
  for i in  множество цветов  кратностью  1 do
  begin
     G1:= G1-V(G1)i;
     G2:= G2-V(G2)i;
  end;
  return (true);
  end.
```

3. ПРОЦЕДУРА СТАБИЛИЗАЦИИ РАСКРАСКИ

Разбиение вершин на классы эквивалентности, внутри которых возможно соответствие изоморфизма обеих графов, является важнейшим шагом при распознавании изоморфизма графов. Существует множество различных инвариантов разбиения [9]. Далее воспользуемся только инвариантом смежности вершин с вершинами фиксированного цвета, допускающим эффективную реализацию разбиения в процессе работы переборного алгоритма. В качестве основной используем процедуру стабилизации раскраски [1,8, 10] , имеющую временную сложность $O(|E| \log |V|)$.

Предлагаем версию алгоритма получения стабильной раскраски вершин параллельно для обоих сравниваемых графов и в ходе работы проверяющего согласованность раскраски. Воспользуемся следующими обозначениями:

$G = G1 \cup G2$

$V = V(G1) \cup V(G2)$

и соответственно

$P[v] =$ если $V \in V(G1)$ тогда $P(G1)[v]$, иначе $P(G2)[v]$

$P\,j\,i = P(G1)\,j\,i \cup P(G2)\,j\,i$ и т.д.

```
procedure Согласована стабильная раскраска
          returns (Boolean);
Input: граф G=G1∪G2 с начальной раскраской
     P= {V1, ... , Vk} .
Output: значение false если разбиение несогласованно,
     иначе значение      true и стабильная раскраска
       P= {V1, ... , Vz} .
 begin
  ОЖИДАНИЕ : = {1, ... , k} ;
  z:=k;
  while   ОЖИДАНИЕ не пусто    do
  begin
     выбрать и удалить произвольный   i  из
     множества ОЖИДАНИЕ;
     Подразбить все цвета относительно цвета  i ;
     if существуют такие      V(G1) jr in P(G1)ji  и
        V(G2)jr in P(G2)ji ,
        что  | V(G1)jr|=|V(G2)jr| then return (false);
     for j in множество цветов смежных с цветом   i do
     begin
        найти наибольшее подмножество     Vjm in Pji;
        for t:=0 to   Pji  -1 do
        begin
           if r_t=m
              then s:=j;
                        else
                        begin
                           s, z:=z+1;
                           добавить   s  в ОЖИДАНИЕ;
                        end;
                     Vs:=Vjr;
                  end;
               end;
        end;
        return (true);
```

```
  end.
```

procedure.Подразбить все цвета относительно цвета i ;

Input: раскрашенный граф G=(V,E,P)и цвет i.
Output: подразбиение цветов Pj для всех разбитых цветов.

```
begin
 for v in Pi do
 begin
   for w in adj(v) do
    begin
       I:=P[w] ;
       if nextcolour[I]=0 then
       begin
          s:= неиспользованный цвет;
          colourfather[s] :=colourfather[I] ;
          nextcolour[I] :=s;
          nextcolour[s] :=0;
          colourlabel[s] :=colourlabel[I] +1;
       end;
       j:=colourfather[I] ;
       r:=colourlabel(nextcolour[I]);
       перенести вершину w в Vj,r;
    end;
 end;
end.
```

Примечание. На входе определены такие функции:

```
colourfather[i] :=i,
colourlabel[i] :=0,
nextcolour[i] :=0.
```

Временная сложность алгоритма подразбиения всех цветов относительно цвета i равна $O(\sum_{v \in V} |adj(v)|)$. Исходя из этого, нетрудно показать что временная сложность алгоритма стабилизации раскраски равна $O(|E| \log |V|)$, т.е. стабилизация раскраски является "недорогим" средст-

вом разбиения вершин.

4. ИСПОЛЬЗУЕМАЯ СТРУКТУРА ДАННЫХ

Предлагаемая структура данных позволяет получить реализацию алгоритмов с приемлимой временной сложностью и требуемой памятью. Необходимые типы данных определим при помощи средств языка программирования PASCAL [11].

```
vertexindex     : 1. . |V|  ;
edgeindex       : 1 . . 2* |E| ;
colourindex     : 0 . . |E| ;
partitionindex  : 1 . . |V|;
vertices : array[vertexindex] of vertexdata;
edges : array[edgeindex] of vertexindex;
colours : array[clourindex] of colourdata;
partition : array[partitionindex] of vertexindex;
```

Массив edges представляет списки вершин из множества adj, partition - списки вершин, окрашенных в один цвет.

```
vertexdata = record
               firstedge : edgeindex;
               lastedge : edgeindex;
               vertexcolour : colourindex;
               vertexplace : partitionindex
             end;
```

Каждая вершина имеет указатели на начало и конец списка смежных с ней вершин - firstedge и lastedge, цвет - vertexcolour и указатель на место вершины в списке partition-vertexplace.

```
colourdata = record
               firstvertex : partitionindex;
               lastvertex : partitionindex;
               nextcolour : colourindex;
```

```
        colourlabel : integer;
        colourfather : colourindex
    end;
```

Границы цвета в списке vertices указывают firstvertex и lastvetex , nextcolour указывает на подразбиение цвета. Эти ссылки используются в создании Pji ,colourlabel указывает на количество соседей в цвете i (относительно которого произведена перекраска) в множестве Vjt , а colourfather на разбитый цвет j .

5. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

Рассмотренный алгоритм реализован на языке программирования PL/1 для ЭВМ ЕС-1060. Длина программы около 1000 операторов.

Эксперименты проводились на однородных графах 5 и графах схем.

Графы схем, возникают при автоматизации конструкторского проектирования радиоэлектронной и электронно-вычислительной аппаратуры. В основу положим граф, описанный в монографии [2] (модель МК2). В этой модели вершинами долей двухдольного графа являются элементы и соединения соответственно, а ребра представляют контакты, связывающие элементы с соединениями. Элементы, контакты и соединения могут быть раскрашены. Для упрощения работы заменим каждое ребро этой модели простой цепью длины 2, получая при этом трехдольный граф с множеством вершин вершин $V=V_э+Vk+Vc$, где $V_э$ -множество элементов схемы, V_k -множество контактов схемы а V_c -множество соединений схемы.

Такой граф будет иметь только соответствующим образом раскрашенные вершины, и для него правомерно обозначение (V , E , P). На практике такие графы разрежены (средняя степень вершины много меньше максимально возможной, часто даже ограничена константой 4) и имеют большое количество вершин (несколько тысяч).

Получены следующие результаты (в последней графе таблиц указано среднее количество вызовов процедуры Изоморфны)

Неизоморфные регулярные графы степени 5

Число				Время (сек.)			Средн. число вызовов
вершин	ребер	цветов	графов	среднее	минимальное	максимальное	
200	500	I	5	8.50	7.06	I0.I0	20I
400	I000	I	5	29.30	25.23	30.77	40I
600	I500	I	5	5I.44	44.53	60.68	60I
800	2000	I	5	92.69	77.86	97.70	80I
I000	2500	I	5	I45.26	I4I.92	I48.62	I00I

Изоморфные регулярные графы степени 5

Число				Время (сек.)			Средн. число вызовов
Вершин	ребер	цветов	графов	среднее	минимальное	максимальное	
200	500	I	I0	5.58	2.42	8.8I	I09
400	I000	I	I0	I7.82	I.99	30.98	233
600	I500	I	I0	29.02	I3.78	60.79	3I8
800	2000	I	I0	39.76	4.99	92.52	323
I000	2500	I	I0	85.54	6.63	I43.90	548

Неизоморфные графы реальных схем

Число				Время (сек).			Средн. число вызовов
вершин	ребер	цветов	графов	среднее	минимальное	максимальное	
598	672	31	5	1.32	1.13	1.46	1
1179	1454	35	5	1.33	0.99	1.77	1
1614	2038	35	5	1.86	0.93	3.05	1
2031	2498	35	5	2.87	1.90	3.89	1
2829	3546	40	5	3.28	2.01	5.83	1

Изоморфные графы реальных схем

Число				Время (сек)			Средн. число вызовов
вершин	ребер	цветов	графов	среднее	минимальное	максимальное	
598	672	31	5	2.15	2.13	2.17	10
1179	1454	35	5	3.78	3.73	3.85	2
1614	2038	35	5	5.10	5.01	5.19	1
2031	2498	35	5	6.67	6.62	6.71	17
2829	3546	40	5	10.71	10.53	10.96	33

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

I. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительный алгоритмов. М.: Мир, 1976.

2. Бершадский А.М. Применение графов и гиперграфов для автоматизации конструкторского проектирования. Саратов: Саратовский университет, 1983. С. 24-27.

3. Земляченко В.Н., Корнеенко Н.М., Тышкевич Р.И. Проблемы изоморфизма графов // Зап.науч.семинаров ЛОМИ АН СССР, 1982. Т. 118. С. 83-158.

4. Кикуст П.Б. Алгоритм распознавания изоморфизма графов и его применения в идентификации логических сетей // Авт. и выч. техника 1979.. Вып. 4. С. 21-27.

5. Кикуст П.Б. Генератор случайных однородных графов. Рига: ЛГУ им.П.Стучки, 1982. ГосФАП, Инв. № П005605.

6. Харари Ф. Теория графов. М.: Мир, 1973.

7. Babai L., Erdos P., Selkow P. Randon graph isomorphism // SIAM J. COMP. 1980. N.3, Vol.9. P. 628-635.

8. Cardon A., Crochemore M. Partitioning a grapg in O (A log V). // Theor.Comp.Sci., 1982, Vol.19. P. 85-98.

9. Cornelein D.G. Kirpatrik D.G. A theoretical analysis of various heuristics for the graph isomorphism problem // SIAM J. Comp. 1980. Vol.9. N.4. P. 247-259.

10. Gries D. Describing an algoritm by Hopcroft // Acta Inf., 1973. Vol. 2. P. 97-109.

11. Kocay W.L. Abstract data types and graph isomorphism // J. of Comb. Inf. and Syst. Sci., 1984. Vol. 9. N.4. P. 247-259.

12. Kubo N., Shirakawa I., Ozaki H. On efficiency improvement for algoritm of testing graph isomorphism // Trans.Inst.Electron. and Com.Eng.Jap., 1978 Vol. 11 P.1099-1105.

13. Luks R.M. Isomorphism of graphs of bounded valence can be tested in polynomial time // Proc. 31st. Symp.Found.Sci., 1980 P.42-89.

АЛГЕБРЫ ОТНОШЕНИЙ В БАЗАХ ДАННЫХ И ОБОБЩЕННЫЕ ЦИЛИНДРИЧЕСКИЕ АЛГЕБРЫ

Я.П.Цирулис
ЛГУ им. П.Стучки

Постановка задачи. По-видимому, первыми работами, в которых явно отмечалась связь между традиционным для реляционных баз данных понятием алгебры отношений, восходящем к Кодду, и хорошо изученным в алгебраической логике понятием цилиндрической алгебры, являются сообщение [1] и его более полная версия [2]. Позже автором в [3] была описана модель базы данных, в которой в качестве алгебры отношений уже непосредственно используется цилиндрическая алгебра множеств. Эта идея была подсказана работой [4], в которой для подобной цели привлекались полиадические алгебры. Не вдаваясь здесь в подробное обсуждение, в двух словах разъяснить причину и правомерность появления цилиндрических алгебр в теории баз данных можно следующим образом: с одной стороны, т. наз. реляционное исчисление с переменными по доменам (см.[5]) близко к исчислению предикатов первого порядка с равенством, а с другой - цилиндрические алгебры возникли в результате алгебраизации последнего и соотносятся с ним примерно так же, как булевы алгебры с исчислением высказываний.

При описании реляционной модели данных с каждой переменной-атрибутом x из некоторого, фиксированного для этой

модели множества X связывают ее область значений - непустое множество (домен) U_x, которое воспринимается как множество данных соответствующего сорта. Все эти домены образуют многосортное множество $U := (U_x : x \in X)$, а в том случае, когда они организованы в некоторую структуру посредством выполнимых над данными операций, говорят даже про алгебру данных U. Однако обычно (в частности, и во всех упомянутых выше работах), обсуждая вопросы, связанные с алгебрами отношений, операции не учитываются. Например, в реляционном исчислении, о котором говорилось выше, аргументами отношений и операторов сравнения являются только переменные и их значения. Но вместо них могли бы стоять и составные арифметические выражения (термы) - это увеличивало бы выразительность языка запросов (ср.[5], 4.3). Отметим еще, что в общем случае в языке могут присутствовать не только арифметические термы, но также и термы других сортов. Если при этом среди используемых сортов имеется и сорт булевозначных величин и доступны обычные логические операции на соответствующей домене, то в принципе достаточно из всех операторов сравнения оставить в языке лишь равенство.

Стандартное понятие цилиндрической алгебры тоже ориентировано на языки без функциональных символов и составных термов. Поэтому достаточно естественно возникает задача о надлежащем обобщении этого понятия. Один способ такого обобщения и предлагается в этой статье. Для простоты мы все же ограничиваемся случаем, когда все атрибуты, а значит, и данные имеют один и тот же сорт. Отметим лишь еще, что аналогичную задачу можно ставить и для других видов алгебр отношений. Для полиадических алгебр она решалась в [6].

Цилиндрические алгебры. Пусть X - фиксированное в дальнейшем множество - множество переменных. Будем называть кванторной алгеброй всякую алгебру $(A, C_x)_{x \in X}$, где A - булева алгебра, а каждое C_x - т. наз. квантор на A, т.е. одноместная операция, обладающая свойствами

$$C_x 0 = 0, \quad a \leq C_x a, \quad C_x(a \wedge C_x b) = C_x a \wedge C_x b,$$

причем разные кванторы перестановочны: $C_x C_y a = C_y C_x a$. Назовем множество $\Delta a := \{x : C_x a \neq a\}$ базисом элемента a кванторной алгебры.

Если каждый элемент квантерной алгебры имеет конечный базис, то сама алгебра называется локально конечной.

Цилиндрическая алгебра – это квантерная алгебра A, рассматриваемая вместе с некоторой диагональю в ней – т.е. функцией $d: X^2 \to A$, обладающей свойствами

$$\left.\begin{array}{l} d(x,x) = 1, \quad C_z(d(x,z) \wedge d(z,y)) = d(x,y) \text{ для } z \neq x, y, \\ C_z(a \wedge d(z,x)) \wedge C_z(-a \wedge d(z,x)) = 0 \text{ для } z \neq x. \end{array}\right\} \quad (I)$$

Более подробно цилиндрическую алгебру можно представить как алгебру вида $(A, C_x, d_{xy})_{x,y \in X}$, где d_{xy} означает элемент $d(x,y)$. Это определение равносильно стандартному определению цилиндрической алгебры размерности $|X|$ – см. [7,1].

Опишем наиболее важный для нас пример цилиндрической алгебры – алгебру множеств. Выберем непустое множество U и рассмотрим множество U^X всех функций-оценок из X в U. Будем называть носителем множества оценок $\alpha \subseteq U^X$ любое такое множество $Y \subseteq X$, что

$$\forall \varphi, \psi \in U^X (\varphi | Y = \psi | Y \to (\varphi \in \alpha \leftrightarrow \psi \in \alpha)).$$

Таким образом, множество α имеет носитель Y в том и только в том случае, когда она состоит из всевозможных продолжений функций, принадлежащих подходящему подмножеству множества U^Y, и полностью определяется этим подмножеством. Это показывает, что в случае конечного Y α представляет некоторое отношение рода Y (по другой терминологии – в схеме Y). Несколько изменив точку зрения, можно считать, что само α и является этим отношением. (см. [3]).

Множество $\mathcal{P}(U^X)$ всех подмножеств U^X является булевой алгеброй относительно обычных теорико-множественных операций. Для каждого $x \in X$ определим на $\mathcal{P}(U^X)$ еще операцию C_x, полагая

$$C_x a := \{\varphi_u^x : \varphi \in a, u \in U\},$$

где φ_u^x – оценка, отличающаяся от φ разве лишь тем, что она принимает значение u в точке x. Полученная алгебра $Q(U) := (\mathcal{P}(U^X), C_x)_{x \in X}$ является квантерной. Следуя [7], ее элемент будем называть регулярным, если базис его служит также и его носителем. Далее, функция $d: X^2 \to \mathcal{P}(U^X)$, определяемая условием

$$d_{xy} := \{\varphi \in U^X : \varphi x = \varphi y\},$$

является диагональю в $Q(U)$; будем называть ее главной.

Таким образом, мы получили цилиндрическую алгебру $(Q(U), \alpha)$. Ее подалгебры и называются цилиндрическими алгебрами множеств (над U). Одной из них является множество $Rel(U)$ всевозможных отношений любых конечных родов (включая пустой). Она локально конечна, т.к. базис каждого элемента алгебры $Q(U)$, как нетрудно проверить, содержится в любом из его носителей.

Сверхцилиндрические алгебры. Опишем теперь обобщение понятия цилиндрической алгебры, обещанное выше. В этом и в следующем разделах нам в некоторых местах придется существенно пользоваться алгебраической терминологией. Информацию о всех не определенных здесь понятиях общей алгебры можно найти в [7, 8].

Пусть наряду с множеством X зафиксирован также некоторый абстрактный тип данных в смысле [9] (см. там замечание после следствия 3.3). Это значит, что выбрано еще одно множество W, включающее X, а также множество E функций $W \to W$, обладающее следующими свойствами: 1) E замкнуто относительно композиций, 2) каждая функция-подстановка $\alpha: X \to W$ имеет в E в точности одно продолжение $\tilde{\alpha}$, 3) каждый элемент $w \in W$ имеет какое-нибудь конечное множество Y такое, что $\tilde{\alpha} w = w$ для любой подстановки α, тождественной на Y. Пересечение всех таких Y для заданного w обозначим через Δw.

В тривиальном случае, когда $W = X$, E совпадает с X^X, и мы не получим ничего нового по сравнению с предыдущим разделом. Более интересный пример типа данных, связанный темой первого раздела, дает множество термов какого-либо языка исчисления предикатов, определяемое тем или другим множеством Ω символов операций (возможно, и нольместных). В этом случае W фактически является алгеброй сигнатуры Ω (т.наз. алгеброй слов [8]), и в качестве E выступает множество всех ее эндоморфизмов. Тогда Δw – это множество всех входящих в w переменных. (См. также доказательство теоремы I.)

Допустим теперь, что $(A, C_x)_{x \in X}$ – некоторая кванторная алгебра. Сверхдиагональю в ней будем называть всякую функ-

цию $d: W^2 \to A$, удовлетворяющую следующим обобщениям соотношений (I):

$$\left.\begin{array}{l} d(w,w)=1,\ C_2(d(w_1,w_2)\wedge d(z,w)) = d((w/z)w_1, d(w/z\,w_2)) \\ \text{для } z\in\Delta w,\ C_2(a\wedge d(z,w))\wedge C_2(-a\wedge d(z,w))=0 \text{ для } z\notin\Delta w \end{array}\right\} \quad (2)$$

Здесь (w/z) означает функцию из E, принимающую значение w в точке z и тождественную на других переменных. Кванторную алгебру, рассматриваемую вместе с какой-либо сверхдиагональю в ней, будем называть сверхцилиндрической. Очевидно, каждая сверхдиагональ является и диагональю (вернее, продолжением какой-либо диагонали), а поэтому каждая сверхцилиндрическая алгебра - цилиндрической.

Снова в качестве примера приведем алгебру множеств. Пусть опять U - непустое множество, а H - множество функций $W \to U$ такое, что 1) $\varphi\mu \in H$, когда $\mu \in E$ и $\varphi \in H$, 2) каждая оценка φ из U^X имеет в H единственное продолжение $\tilde{\varphi}$. Например, если W - алгебра термов языка (как выше), то U может быть алгеброй данных (т.е. алгеброй, однотипной с W . (См. также доказательство теоремы I.) В любом случае функция $d: W^2 \to \mathcal{P}(U^X)$, определяемая условием

$$d_{vw} := \{\varphi \in U^X : \tilde{\varphi}v = \tilde{\varphi}w\} , \quad (3)$$

является сверхдиагональю в $Q(U)$, и мы получаем сверхцилиндрическую алгебру $(\mathcal{P}(U^X), d)$. Ее подалгебры можно называть сверхцилиндрическими алгебрами множеств (над U). В силу леммы 2.2. из [9] для любого $w \in W$ и всех $\varphi, \psi \in U^X$

$$\varphi|_{\Delta w} = \psi|_{\Delta w} \to \tilde{\varphi}w = \tilde{\psi}w.$$

С учетом этого легко убедиться, что каждое из множеств d_{vw} имеет носитель $\Delta v \cup \Delta w$ и поэтому принадлежит $Rel(U)$. Следовательно, $Rel(U)$ является даже сверхцилиндрической алгеброй множеств.

Сверхдиагонали описанного вида тоже будем называть главными. Понятно, что, в отличие от главной диагонали главных сверхдиагоналей в $Q(U)$ может быть несколько. Для нас важна следующая внутренняя характеристика главных сверхдиагоналей в алгебрах множеств.

ТЕОРЕМА I. Если X бесконечно, то сверхдиагональ d в кванторной алгебре $Q(U)$ является главной тогда и только тогда, когда она является продолжением главной диагонали, и все $d(v,w)$ регулярны.

ДОКАЗАТЕЛЬСТВО. Необходимость условия проверяется просто. Чтобы доказать его достаточность, отметим, что согласно следствию 3.3 из [9] абстрактный тип данных всегда можно превратить в "конкретный" в том смысле, что множество можно превратить в факторалгебру алгебры термов какого-либо языка (т.е. в свободную в себе над X алгебру) так, чтобы E оказалось множеством ее эндоморфизмов. Пусть еще задана сверхдиагональ d в $Q(U)$ с указанными в условии теоремы свойствами. Тогда из приведенной в [10] теоремы вытекает, что множество U можно превратить в алгебру, однотипную с W, так, что при этом d представима в виде (3), где в качестве H выступает множество всех гомоморфизмов из W в U. (Бесконечность множества X существенна для упомянутой теоремы из [10]).

Отметим еще один полезный факт.

ЛЕММА 2. Любое продолжение главной диагонали в алгебре $Q(U)$, являющееся сверхдиагональю в какой-либо из ее подалгебр, оказывается таковой и в ней самой.

По существу нам необходимо лишь показать, что третье из свойств (I) можно распространить на всю алгебру $Q(U)$. Доказательство этого факта довольно громоздко, поэтому мы его здесь опустим.

Теоремы представления. Итак, каждый абстрактный тип данных позволяет определить некоторый класс (многообразие) сверхцилиндрических алгебр. Как мы видели, любая алгебра отношений вида $Rel(U)$ является сверхцилиндрической, и в этом смысле выбранные аксиомы сверхцилиндрических алгебр, прежде всего – аксиомы (2), определяющие понятие сверхдиагонали, можно считать оправданными. В этом разделе будет показано, что никакие дополнительные тождества присоединить к этим аксиомам не требуется (теорема 4).

Цилиндрическую алгебру принято называть представимой, если она изоморфна подпрямому произведению алгебр множеств [7]. В силу следствия 3.1.107 из [7] все эти алгебры можно считать регулярными, т.е. состоящими только из регулярных элементов. Примем такое же определение представимости и для сверхцилиндрических алгебр. Имеет место

ТЕОРЕМА 3. При бесконечном V сверхцилиндрическая алгебра представима тогда и только тогда, когда представима ее цилиндрическая часть.

ДОКАЗАТЕЛЬСТВО. Особого обоснования требует лишь достаточность условия. Если $(A,\mathcal{d})$ - представимая цилиндрическая алгебра, то она изоморфна подпрямому произведению регулярных алгебр множеств $(A_i,\mathcal{d}_i)_{i\in I}$ с каноническими проекциями $\varepsilon_i: A\to A_i$. Если теперь диагональ $\mathcal{d}$ каким-либо образом продолжена до сверхдиагонали $\tilde{\mathcal{d}}$, то для каждого i функция $\tilde{\mathcal{d}}_i := \varepsilon_i\tilde{\mathcal{d}}$ является продолжением главной диагонали $\mathcal{d}_i$ (равной $\varepsilon_i\mathcal{d}$). Учитывая, что A_i - полный образ A, нетрудно проверить, что $\mathcal{d}_i$ - сверхдиагональ в A_i, так что согласно лемме 2 она будет сверхдиагональю и в соответствующей алгебре $Q(U_i)$. В то же время в силу выбора A_i все элементы вида $\tilde{\mathcal{d}}_{(v,w)}$ регулярны, значит, в силу теоремы I $\mathcal{d}_i$ - главная сверхдиагональ в A_i. При этом сверхцилиндрическая алгебра $(A,\tilde{\mathcal{d}})$, очевидно, остается подпрямым произведением алгебр $(A_i,\tilde{\mathcal{d}}_i)$ с теми же проекциями ε_i. Таким образом, $(A,\tilde{\mathcal{d}})$ тоже представима, и теорема доказана.

Естественно называть алгеброй отношений над множеством U произвольную подалгебру сверхцилиндрической алгебры $\mathfrak{Rel}(U)$.

Мы теперь в состоянии дать абстрактную характеристику алгебр отношений. Напомним, что некоторая алгебра называется простой, если любой нетривиальный ее гомоморфный образ изоморфен самой алгебре.

ТЕОРЕМА 4. Если V бесконечно, то сверхцилиндрическая алгебра тогда и только тогда изоморфна алгебре отношений, когда она проста и локально конечна.

ДОКАЗАТЕЛЬСТВО. Известно, что все локально конечные регулярные цилиндрические алгебры множеств простые ([II], теорема I.I0). Поэтому нам остается рассмотреть лишь достаточность условия. Допустим, что сверхцилиндрическая алгебра $(A,\mathcal{d})$ обладает обоими указанными свойствами. Так как при бесконечном X любая локально конечная алгебра представима ([7], теорема 3.2.8), для алгебры $(A,\mathcal{d})$ имеется гомоморфизм ε_i на регулярную алгебру множеств $(A_i,\mathcal{d}_i)$ над U_i (см. предыдущее доказательство). Тогда ввиду своей простоты алгебра $(A,\mathcal{d})$ изоморфна $(A_i,\mathcal{d}_i)$. Значит, алгебра $(A_i,\mathcal{d}_i)$ локально конечна,

поэтому ввиду ее регулярности каждый элемент A_i имеет конечный носитель, т.е. принадлежит $Rel(U_i)$.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

I. Imieliński T., Lipski W. The relational model of data and cylindric algebras // ICS PAS Reports. 1981. Nr. 446.

2. Imieliński T., Lipski W. The relational model of data and cylindric algebras // J. Comp. Syst. Sci. 1984. V. 28. P. 80-102.

3. Цирулис Я.П. Модель базы данных с недетерминированными переходами // Теория алгоритмов и программ. Рига: ЛГУ им. П.Стучки, 1986. С. 87-95.

4. Плоткин Б.И. Алгебраическая модель базы данных-автомата // Латв. мат. ежегодник. Рига: Зинатне, 1983. Вып. 27. С. 216-232.

5. Ульман Дж. Основы систем баз данных. М.: Финансы и статистика, 1983.

6. Cirulis J. Generalizing the notion of polyadic algebra // Bull. Sect. Log. 1986. V. 15. Nr. 1. P. 2-9.

7. Henkin L., Monk J.D., Tarski A. Cylindric algebras. Amst., N.Y., Oxford: North-Holland. Part I. 1971, 1985. Part II. 1985.

8. Кон П. Универсальная алгебра. М.: Мир. 1968.

9. Цирулис Я.П. Абстрактное описание типов данных и многообразий алгебр данных // Алгебра и дискретная математика: Теоретические основы математического обеспечения ЭВМ. Рига: ЛГУ им. П.Стучки, 1986. С. 131-144.

10. Цирулис Я.П. Сверхдиагонали универсальных алгебр // XIX Всес. алгебр. конф.: Тезисы сообщ. Львов, 1987. Ч. II. С. 306.

11. Henkin L., Tarski A. Cylindric algebras // Proc. symp. pure math. V. II: Lattice theory. Provid., North Island: Amer. Math. Soc., 1961. P. 83-113.

РАСПОЗНАВАНИЕ ПЛАНАРНЫХ ИЕРАРХИЧЕСКИХ ОБРАЗОВ НА ОСНОВЕ ПРЕОБРАЗОВАНИЙ ПЛОСКОСТИ

П.Б.Кикуст
ВЦ при ЛГУ им.П.Стучки

Процесс восприятия окружающей обстановки представляется естественным разбить на следующие чередующиеся этапы:

- направление внимания на ограниченный участок окружения,
- распознавание объектов, имеющихся в выделенном участке.

Типичным примером такого процесса является чтение человеком письменного текста. В качестве окружающей обстановки здесь выступает плоская сцена - лист бумаги с письменными знаками на нем. Аналогичными примерами являются анализ фотоснимков и оценивание ситуации на конвейерной ленте с целью отыскания нужной детали небольшой толщины.

В каждом из направлений, характеризуемых приведенными примерами, достигнуты определенные успехи при создании автоматизированных систем обработки изображений [6,7,9, II,I2,I5,I6,I7]. Однако при этом ярко прослеживается как акцент на узкую специализированность решений [6,9,II,I2, I5,I6], так и отсутствие крупных общих подсистем в части собственно распознавания. С другой стороны, общепризнанность структурно-лингвистического подхода [I,2] предрасполагает и к разработкам общего назначения [I7].

Предметом настоящей работы является программная реализация второго из названных во вступлении этапов восприятия в случае плоских составных сцен из объектов, имеющих иерархическое строение, возможно, соприкасающихся или частично перекрывающих друг друга. Связный рукописный текст, аэрофотоснимки или набор нетолстых деталей на конвейерной ленте как раз являются примерами таких сцен.

Предложенное решение согласуется с точкой зрения

происходит в условиях различных взаимных расположений системы и окружающей среды, то в такой системе должна быть заложена инвариантность распознавания по отношению к преобразованиям изображения распознаваемого объекта в соответствии с его расположением. Основным классом таких преобразований следует считать проективные преобразования плоскости, в приложениях ограничиваясь такими его подклассами как афинные преобразования, движения и т.п. [4,5,13].

С другой стороны, наличие у распознаваемого объекта относительно автономных частей (например, отдельные штрихи или их совокупности у письменных знаков) влечет возможность их автономного расположения на анализируемом изображении вплоть до отсутствия. Поэтому мы требуем от зрительной системы также инвариантности распознавания по отношению и к вариациям формы этого вида.

Целью настоящей работы является построение универсальной вычислительной процедуры, удовлетворяющей выдвинутым требованиям, т.е. определяющей все вхождения заданного образа в изображение и обеспечивающей инвариантность распознавания как по отношению к допустимым преобразованиям плоскости, так и по отношению к вариациям формы, обусловленным возможностью автономного расположения составных частей образа. Мы убеждены, что именно такая процедура в качестве базового звена должна быть включена в любую систему распознавания изображений, а требование универсальности заставляет нас воздержаться от преждевременного принятия специализированных решений эвристического характера. Именно это отличает наш подход от, например, [6,11,15,16], где на передний план выступает чисто прикладной аспект решения поставленных там задач.

Таким образом, основная идея работы оказывается предельно простой и заключается в привлечении перебора взаимных расположений частей образа. Хотя такой логически простой прием практически не осуществим на традиционных ЭВМ, он является самым естественным для многопроцессорных вычислительных устройств и поэтому заслуживает тщательного изучения. Далее следует изложение некоторых важных

[10,14] и для практического использования предполагает наличие специализированных вычислительных устройств параллельного действия. Однако с целью демонстрации состоятельности выдвинутых идей приведена также версия программы для традиционной ЭВМ и результаты ее работы на небольшом тестовом примере.

I. ПРИНЦИП РАСПОЗНАВАНИЯ

Выделим четыре уровня зрительного восприятия:

- формирование и первичная обработка изображения ограниченного участка сцены,
- поиск на изображении непроизводных элементов распознаваемых образов,
- распознавание и локализация образов на основе найденных непроизводных элементов,
- обработка полученной информации и принятие решения о направлении внимания на следующий участок сцены.

Ключевым для нас является третий уровень, рассмотрению которого, согласно вышесказанному, посвящена настоящая работа. В необходимой степени приведем также детали, касающиеся и первых двух уровней.

Мы ограничимся определением функциональной процедуры распознавания отдельного образа. Однако от процедуры потребуем вычисления всех вхождений заданного образа в выделенный участок сцены.

Выделенный ограниченный участок плоской сцены будем называть изображением. Образ зададим при помощи эталонных изображений его непроизводных элементов (атомарных образов) и схемы его иерархической структуры. Под вхождением образа P в изображение I будем понимать существование такого преобразования плоскости T из множества допустимых, что изображение $T(P)$ с определенной точностью совпадает с некоторой частью изображения I. Другими словами, мы предполагаем инвариантность распознавания по отношению к множеству допустимых преобразований плоскости.

Так как функционирование реальной зрительной системы

технических деталей, появляющихся в ходе первого продвижения в данном направлении.

2. ОСНОВНЫЕ ПОНЯТИЯ

Определим функции, вычисление которых необходимо для решения вопроса о наличии заданного образа в изображении. В данном разделе, во избежание излишней громоздкости изложения, рассмотрим недетерминированные функции. Определения будем строить в стиле фуниционального программирования [8].

Иерархическую структуру рассматриваемых образов зададим следующим правилом:

образ :: = атомарный образ | список образов.

По отношению к атомарному образу будем считать заданной булеву функцию is-atomar-pattern и недетерминированную функцию ap-transf (P) = преобразование T, для которого при условии, что P - атомарный образ, T(P) содержится в заданном изображении, или значение none , если такого преобразования не существует.

Далее потребуются следующие функции общего характера.

none-del (x) = список элементов списка x, отличных от none.

length(x) = количество элементов списка x.

sum(x) = сумма элементов числового списка x.

map(f,(x_1 x_2 ... x_n)) = ($f(x_1)$ $f(x_2)$ ... $f(x_n)$).

$\lambda(x_i)$ $f(x_1,x_2,\ldots,x_n)$ = функция, полученная из функции f фиксированием значений всех аргументов, кроме x_i.

Функции для работы с преобразованиями.

is-near-transf (T1,T2) = число 0 или 1 , сопоставленное паре преобразований T1,T2.

res-transf (TL) = преобразование, сопоставленное списку преобразований TL.

```
p-transf(TL) = if length(none-del(TL))/length(TL) > Q1  &
                  sum(map( λ(x) is-near-transf
                                 (x,res-transf(none-del(TL))),
                           none-del(TL)))/
                  length(none-del(TL)) > Q2
                  then res-transf(none-del(TL))
                  else none
```

Ответ на вопрос о вхождении и местонахождении образа P в изображении дает результат вычисления следующей функции.

```
present(P) = if is-atomar-pattern(P)
             then ap-transf(P)
             else p-transf(map(present,P)).
```

Следует признать, что независимость функции p-transf от обрабатываемого образа является преднамеренным упрощением положения вещей на данной стадии исследований.

Введенный набор функций предполагает большой объем вычислений. Однако недетерминизм, заложенный в основе этих функций, обусловливает высокую степень параллелизма при их выполнении и тем самым открывает путь для их воплощения в реальных устройствах распознавания изображений. Сказанное подкрепляет также развитие языка программирования ПРОЛОГ [3], поддерживающего недетерминированные вычисления, и на котором наши функции записываются практически без изменений.

В качестве примера приведем последнее из них:

```
present(P,T):-is-atomar-pattern(P),ap-transf(P,T).
present(P,T):-map(present,P,TL),p-transf(TL,T).
```

В ответ на запрос ?-present (P,T) при заданном образе P, для переменной T будут вычислены все соответствующие значения.

3. ВЫЧИСЛИТЕЛЬНАЯ МОДЕЛЬ

Проверка работоспособности предложенного набора функций осуществлялась традиционными средствами программирования. Для этой цели состав набора был несколько расширен.

ap-transf-list (P) = список всех преобразований T, для которых, при условии, что P - атомарный образ, T(P) содержится в заданном изображении, или список (none), если такого преобразования не существует.

cartprod ((x$_1$ x$_2$ ... x$_k$)) = список элементов декартового произведения $x_1 \times x_2 \times \ldots \times x_k$ при условии, что $x_1, x_2, \ldots, x_k$ - непустые списки.

```
none-reduction(x) = if length(none-del(x))=0
                      then (none)
                      else none-del(x)
```

Функция present видоизменялась следующим образом.

```
present(P) = if is-atomar-pattern(P)
             then ap-transf-list(P)
             else none-reduction
                  (map (p-transf,
                        cartprod(map(present,P)))).
```

При осуществлении конкретных вычислений мы в качестве допустимых преобразований ограничивались движениями плоскости (исключая зеркальное отображение). Такое преобразование однозначно определяется списком (u p q), где u - угол поворота, а p,q - величина сдвига по x и y осям соответственно.

При этих предположениях окончательную форму приобретают следующие функции.

```
is-near-transf((u1 p1 q1)(u2 p2 q2)) =
   = if sin(|u1-u2|) < A & |p1-p2| < B & |q1-q2| < B
     then 1 else 0.
```

res-transf((u$_1$p$_1$q$_1$)(u$_2$p$_2$q$_2$)...(u$_k$p$_k$q$_k$)) =
= (среднее арифметическое от величин $u_1, u_2, \ldots, u_k$

среднее арифметическое от величин $p_1, p_2, \ldots, p_k$

среднее арифметическое от величин $q_1, q_2, \ldots, q_k$),

где при вычислении среднего от углов, учитываются свойства периодичности величин последних.

Для окончательного уточнения функции ap-transf-list необходимо сначала определить представление изображения, в том числе и эталонных изображений атомарных образов:

код изображения ::= список примитивных элементов изображения,

примитивный элемент изображения ::= (a x y).

Наличие в коде изображения примитивного элемента (a x y) означает, что на изображении вблизи точки с координатами x,y проходит контурная или скелетная линия, касательная к которой в этом районе образует с осью абсцисс угол, близкий a . Такой код легко построить при помощи традиционных средств выделения контурных и скелетных линий на стадии предварительной обработки изображения.

Преобразования осуществим посредством множества опорных точек, с которыми совместим центры атомарных образов. Пусть XL и YL - списки абсцисс и ординат опорных точек, а UL - список допустимых углов поворота при преобразованиях,

Далее следует заключительный набор функций.

max (x) = наибольший элемент числового списка x.

is-near $((a_1\ x_1\ y_1)(a_2\ x_2\ y_2))$ =

$= \underline{\text{if}}\ (x_1-x_2)^2 + (y_1-y_2)^2 < R^2\ \&\ \sin^2(a_1-a_2) < S^2$

then 1 else 0.

is-in-image(E) = max(map(λ(x) is-near(E,x),I)),

где E - примитивный элемент атомарного образа, I - код заданного изображения.

include-number(T,P) = sum(map(is-in-image,

map(λ(x)transf-apply(T,x),

P))).

transf-apply ((u p q) (a x y)) =

= (a+u x.cos(u)-y.sin(u)+p x.sin(u)+y.cos(u)+q).

```
transf-accept(T,P)=if include-number(T,P)/length(P) > Q3
                   then T else none.
xcentre(P) = (xmin + xmax)/2          , где xmin, xmax -
```

наименьшее и наибольшее значения абсцисс примитивных элементов, составляющих атомарный образ P.

```
ycentre (P)    = аналогично для ординат.
transform(P,(u x y)) = (u x-xcentre(P) y-ycentre(P)).
ap-transf-list (P) = none-reduction
                     (map (λ(x) transf-accept(x,P),
                          map( λ(x) transform (P,x),
                               cartprod ((UL XL YL))))).
```

Этим завершено построение одного варианта функциональной процедуры распознавания. Остается лишь выбрать конкретные значения для пороговых величин Q_1, Q_2, Q_3, A, B, R, S и элементы списков UL, XL, YL.

Испытания полученной программы выявили наличие больших скоплений близких преобразований, излишне дублирующих друг друга. Мы приняли решение о введении дополнительной легко определяемой функции transf-list-reduction, которая находит центры таких скоплений и только их оставляет в списке преобразований.

Тем самым окончательный вид функции present оказывается следующим.

```
present (P) = transf-list-reduction
              (if is_atomar_pattern (P)
                 then ap_transf_list (P)
                 else none-reduction
                      (map(p_transf,
                           cartprod
                           (map(present,P))))).
```

4. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ

После стадии предварительной обработки, на которой выделялись и сглаживались контурные и скелетные линии изображения, последнее было приведено к квадратной области плоскости, ограниченной прямыми x=0, x=30, y=0, y=30.

Для распознавания полагалось:

UL = (-24 0 24) (углы даны в градусах),

XL=YL=(2 5 8 11 14 17 20 23 26 29),

$Q_1=Q_2=Q_3=0.75$,

R=2, S=0.21, A=0.17, B=2.3.

Рис.I. представляет типичный образец изображения, на котором требовалось распознать буквы "Р" и "К", эталонные изображения которых даны на рис.2. Одинаковыми цифрами на рис.2.

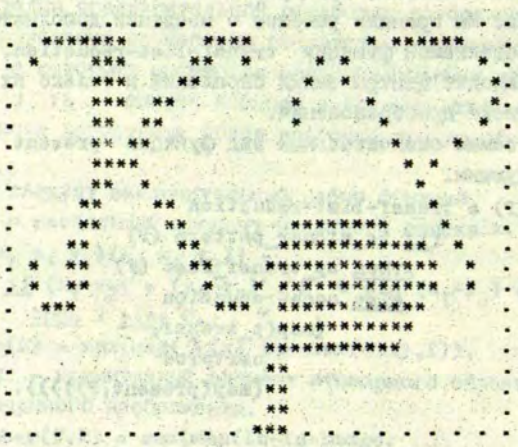


Рис.I. Образец распознаваемого изображения.

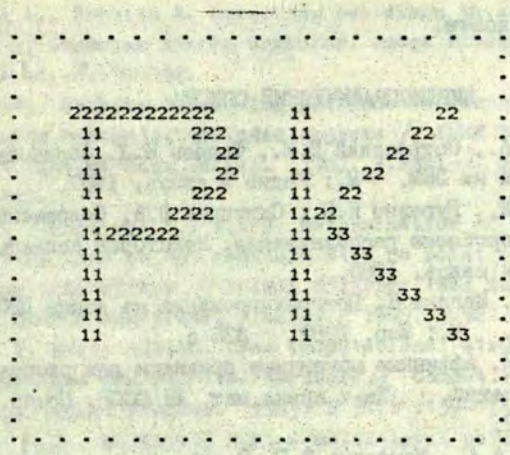


Рис.2. Эталонные изображения букв "Р" и "К".

изображены атомарные образы. Иерархическое строение буквы "Р" задавалось списком (1 2), а строение буквы "К" списком (1 (2 3)).

С целью явного указания местонахождения образа на изображении соответствующие преобразования были приведены к виду (u x y) , где u - угол поворота, а x,y - координаты точки, в которую перемещается центр тяжести эталонного изображения образа. Обработка показанного на рис.I изображения дала следующие результаты:

для буквы "Р" - (0 17.6 8.5), (24 25.4 24.2),
для буквы "К" - (-24 9.9 20.8).

Разумеется, легко построить изображения, на которых предложенные выше простые функции не будут работать адекватно. Для усовершенствования процедуры, в первую очередь требуется индивидуализировать значения пороговых величин для каждого образа и его составных частей, что снимет ранее отмеченную независимость функции p_transf от обрабатываемого образа. Необходимо также построить более изощренные функции is_near_transf и res_transf , определенные и для преобразований, более общих, чем просто

движения плоскости.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

I. Бутаков Е.А., Островский В.И., Фадеев И.Л. Обработка и изображений на ЭВМ. М.: Радио и связь, 1987.

2. Горелик А.Л., Гуревич И.Б., Скрипкин В.А. Современное состояние проблемы распознавания: Некоторые аспекты. М.: Радио и связь, 1985.

3. Клоксин У., Меллиш К. Программирование на языке ПРОЛОГ: Пер.с англ. М.: Мир, 1987. 336 с.

4. Кугушев Е.И. Аффинные моментные признаки для распознавания изображений. Ин-т прикл.мат. АН СССР. Препр. 1987. № 132.

5. Майстренко А.А., Маштелир В.П. Распознавание изображений при ортогональных проективных преобразованиях //Цифровые методы в управлении, радиолокации и связи: Межвуз. сб. Свердловск; УПИ им.С.М.Кирова, 1986. С.65-71.

6. Страхович Э.В. Распознавание рукописных знаков с помощью лингвистических методов // Информатика и вычислительная техника: Тезисы докладов всесоюзного семинара молодых ученых и специалистов, Звенигород, 11-18 апреля 1986 г. М.: Наука. 1986. С. 188-189.

7. Фролов Ф.В., Гуцу В.Г. Иерархические системы распознавания рукописных цифр Известия АН МССР, Сер.Физ.-техн. и мат.н. 1985. № 2. С.12-17.

8. Хендерсон П. Функциональное программирование. Применение и реализация. М.: Мир, 1983.

9. Ayache N., Fangeras O.D. HYPER: A New Approach for the Recognition and Positioning of Two-dimensional Objects. IEEE Trans.Pattern Anal.Machine Intell. 1986. V.PAMI-8, № 1, P.44-54.

10. Feldman J.A. Connectionist models and parellelism in high level vision. Comput.Vision Graphics, Image Process. 1985. N 31. P.178-200.

11. Huertas A., Nevatia R. Detecting buildings in aerial images // Computer Vision Graphics, Image Process. 1988. N 41. P.131-152.
12. Koch M.W., Kashyap R.L. Using poligons to recognize and locate partially occluded objects // IEEE Trans. Pattern anal.Machine Intell. 1987. V.PAMI-9., N 4. P.483-494.
13. Orr J.A., Cyganski D., Vaz R. Determination af affine transforms from object contours with no point correspondence information // ICASSP 85: Proc.IEEE Int.Conf. Acoust.Speech and Signal Process. 1985. P.921-924.
14. Poggio T. Early vision: from computational structure to algorithms and parallel hardware // Comput.Vision Graphics, Image Process. 1985. N 31. P.139-155.
15. Srihari S.N., Bozinovic R.M. A multi-level perception approach to reading cursive script // Artificial Intelligence. 1987. N 33. P.217-255.
16. Tsui H.T., Chan M.H. Recognition of partially occluded two-dimensional objects // IEE Proceedings, 1987. V. 134. Part E. N 1. P.9-16.
17. Wallace A.M. An informed strategy for matching models to images of fabricated objects // Pattern Recognition, 1987. V .20. N 3. P.349-363.

МИНИМАЛЬНОЕ ЧИСЛО ЗАДАВАЕМЫХ ВОПРОСОВ В ДЕРЕВЬЯХ РЕШЕНИЙ

Р.В.Фрейвалд
ВЦ при ЛГУ им. П.Стучки

Для булевых функций давно рассматривается такая мера сложности как число существенных переменных.

Переменная x_i для функции $f(x_1,\dots,x_n)$ называется существенной, если существует такой набор значений переменных $(a_1,\dots,a_{i-1},a_{i+1},\dots,a_n)$, что $f(a_1,\dots,a_{i-1},0,a_{i+1},\dots,a_n) \neq f(a_1,\dots,a_{i-1},1,a_{i+1},\dots,a_n)$.

Число существенных переменных является очень грубой мерой сложности. В /1/ была рассмотрена близкая по духу, но немного более тонкая мера сложности: число задаваемых вопросов в дереве решений.

Вычисление фиксируемой булевой функции при помощи дерева решений означает, что в "черном ящике" скрыт набор значений всех переменных этой функции. Разрешается задавать вопросы типа : "чему равно x_{17} ?", "чему равно x_{23} ?" и т.д. При формулировании следующего вопроса можно учитывать ответы на предыдущие вопросы. Новая мера сложности "число задаваемых вопросов" выражает максимум числа подряд задаваемых вопросов по всевозможным наборам значений аргументов функции.

Может показаться, что число задаваемых вопросов всегда равно числу существенных переменных. Однако это не так. Например, рассмотрим следующую последовательность булевых функций $\{f_k\}_{k=1,2,3,\dots}$. Функция f_k вычисляется заданием k вопросов. Первый вопрос всегда "чему равно x_1 ?" Если i вопрос (при $i<k$) имеет вид "чему равно x_ℓ ?", то при ответе 0 на этот вопрос следующий вопрос относится к $x_{2\ell}$, а при ответе 1 следующий вопрос относится к $x_{2\ell+1}$. После того, когда задан k-й вопрос, другие вопросы уже не задаются, и значение функции f_k равно ответу на последний вопрос. Очевидно справедливо

ПРЕДЛОЖЕНИЕ 1. (1) Функция f_k имеет 2^K-1 существенных переменных.

(2) Функция f_k вычисляется детерминированным деревом решений заданием к вопросов.

ПРЕДЛОЖЕНИЕ 2. Если булева функция f вычисляется заданием к вопросов, то f имеет не более чем 2^K-1 существенных переменных.

ДОКАЗАТЕЛЬСТВО. Рассмотрим дерево решений, т.е. дерево, в корень которого помещен первый вопрос, из каждой вершины которого выходят ровно две дуги, - они соответствуют ответам "0" и "1". Очевидно, такое к-ярусное двоичное дерево не может содержать больше чем 2^K-1 вершин. □

Итак, для вычисления булевой функции с n существенными переменными детерминированным деревом решений необходимо не менее чем $\log_2 n$ вопросов. Ниже будет показано, что для недетерминированных деревьев решений это число вопросов может быть уменьшено.

Уточним, что недетерминированное дерево решений может содержать несколько начальных вершин, и из любой вершины может выходить несколько дуг, соответствующих ответу "0", и несколько дуг, соответствующих ответу "1". Недетерминированность алгоритма выражается в том, что существует возможность произвольно выбрать в дереве тот или иной следующий вопрос.

Будем говорить, что недетерминированным деревом решений булеву функцию f можно вычислить с s вопросами, если для каждого набора значений переменных $\alpha = (\alpha_1, \ldots, \alpha_n)$ существует такая реализация работы алгоритма, при которой задается не более чем s вопросов и выдается ответ $f(\alpha_1, \ldots, \alpha_n)$, и, кроме того, ни при какой реализации работы алгоритма не выдается ложный ответ.

ТЕОРЕМА 3. Существует такая последовательность булевых функций $\{g_k\}_{k=1,2,3,\ldots}$, что:

(1) функция g_k вычисляется недетерминированным деревом решений заданием к+1 вопросов;

(2) функция g_k имеет $\binom{2k}{k}+2k$ существенных переменных.

КОММЕНТАРИЙ. Если обозначить $\binom{2k}{k}+2k$ через n, то видно, что функция g_k с n переменными может быть вычислена недетерминированным деревом решений заданием $\frac{1}{2}\cdot\log_2 n + o(\log_2 n)$ вопросов.

ДОКАЗАТЕЛЬСТВО. Функция g_k определяется так. Если среди первых 2к переменных этой функции не менее чем к+I принимают значение I, то значение функции равно I. Если среди первых 2к переменных этой функции не менее чем к+I принимают значение 0, то значение функции равно 0. Остается определить функцию на тех наборах значений, где среди первых 2к переменных к принимают значение I и к - значение 0. Таких наборов ровно $\binom{2k}{k}$. Каждому такому набору поставлена в соответствие своя переменная x_i $(i > 2k)$. Тогда значение функции g_k равно значению этой переменной x_i.

Для недетерминированного вычисления значения функции, равной I, надо догадаться спросить либо значения тех к+I среди первых 2к переменных, которые равны I, либо значения тех к среди первых 2к переменных, которые равны I, и значение x_i, которое должно оказаться равным I. Для недетерминированного вычисления значения функции, равной 0, надо догадаться спросить либо значения тех к+I среди первых 2к переменных, которые равны 0, либо значения тех к среди первых 2к переменных, которые равны 0, и значение x_i, которое должно оказаться равным 0.

Автору неизвестно, можно ли для какой-то последовательности булевых функций в теореме 3 добиться оценки меньшей чем $\frac{1}{2}\log_2 n + o(\log_2 n)$.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Manber U., Tompa M. The complexity of problems on probabilistic, nondeterministic and alternating decision trees //Journal of the ACM, 1985. V.32. P.710-732.

РАСПОЗНАВАНИЕ ЯЗЫКОВ БЕСКОНЕЧНЫХ СЛОВ НА МАШИНАХ ТЬЮРИНГА

Д.Я.Тайминя
ЛГУ им. П. Стучки

Теория автоматов, перерабатывающих бесконечные слова, возникла в шестидесятых годах в работах Бюхи /1/ и Макнотона /2/ в рамках математической логики. Рассмотрение бесконечных слов на первый взгляд может показаться слишком абстрактным. Однако бесконечную работу автомата естественно рассматривать, когда мы хотим исследовать системы, остановка которых не предусмотрена. Примером таких систем могут быть операционные системы ЭВМ.

Машина Тьюринга для работы с бесконечными словами отличается от машины Тьюринга, рассматриваемой обычно, лишь тем, что у нее с самого начала для записи аргумента использована вся бесконечная лента, а также тем, что вместо принимающих и отвергающих состояний имеются принимающие и отвергающие множества состояний. Для определения того, принимается ли данное бесконечное входное слово надо рассмотреть множество всех тех внутренних состояний машины, которые при работе машины на этом бесконечном входном слове повторяются бесконечно много раз. Слово принимается (или отвергается), если это множество входит в список принимающих (отвергающих) множеств состояний.

Охарактеризуем языки бесконечных слов, распознаваемые на детерминированных и недетерминированных машинах Тьюринга.

Классом Σ_1 называется класс всех отношений $P(x_1, \dots, x_k, y_1, \dots, y_\ell)$, представимых в виде

$$P(x_1, \dots, x_k, y_1, \dots, y_\ell) = (\exists y_{\ell+1})(R(x_1, \dots, x_k, y_1, \dots, y_\ell, y_{\ell+1})),$$

где R - рекурсивное отношение относительно $(\Sigma^\omega)^k \times (\Sigma^*)^{\ell+1}$.

Классом Π_1 называется класс всех отношений $P(x_1, \dots, x_k, y_1, \dots, y_\ell)$, представимых в виде

$P(x_1,\dots,x_k,y_1,\dots,y_e) = (\forall y_{e+1})(R(x_1,\dots,x_k,y_1,\dots,y_e,y_{e+1}),$
где R - рекурсивное отношение относительно $(\Sigma^\infty)^k \times (\Sigma^*)^{e+1}$.

Классом Σ_{n+1} называется класс всех отношений $P(x_1,\dots,x_k,y_1,\dots,y_e)$, представимых в виде

$P(x_1,\dots,x_k,y_1,\dots,y_e) = (\exists y_{e+1})(R(x_1,\dots,x_k,y_1,\dots,y_e,y_{e+1}),$
где R - отношение класса Π_n относительно $(\Sigma^\infty)^k \times (\Sigma^*)^{e+1}$.

Классом Π_{n+1} называется класс всех отношений $P(x_1,\dots,x_k,y_1,\dots,y_e)$, представимых в виде

$P(x_1,\dots,x_k,y_1,\dots,y_e) = (\forall y_{e+1})(R(x_1,\dots,x_k,y_1,\dots,y_e,y_{e+1}),$
где R - отношение класса Σ_n относительно $(\Sigma^\infty)^k \times (\Sigma^*)^{e+1}$.

Классом Σ_1^1 называется класс отношений $P(x_1,\dots,x_k,y_1,\dots,y_e)$, представимых при некотором $s \geq 0$ в виде

$$P(x_1,\dots,x_k,y_1,\dots,y_e) = (\exists x_{k+1})(\exists y_{e+1})(\forall y_{e+2})(\exists y_{e+3})\dots(Q y_{e+s})$$
$$R(x_1,\dots,x_k,x_{k+1},y_1,\dots,y_e,y_{e+1},\dots,y_{e+s}),$$

где R - рекурсивное отношение относительно $(\Sigma^\infty)^{k+1} \times (\Sigma^*)^{e+s}$, а квантор $(Q y_{e+s})$ - квантор существования при s нечетном и квантор всеобщности s четном или в виде

$$P(x_1,\dots,x_k,y_1,\dots,y_e) = (\exists x_{k+1})(\forall y_{e+1})(\exists y_{e+2})(\forall y_{e+3})\dots(Q y_{e+s})$$
$$R(x_1,\dots,x_k,x_{k+1},y_1,\dots,y_e,y_{e+1},\dots,y_{e+s}).$$

По теореме Клини о нормальной форме (см. /3/, упражнение I6-I0) каждое отношение P из Σ_1^1 можно представить в виде

$$P(x_1,\dots,x_k,y_1,\dots,y_e) = (\exists x_{k+1})(\forall y_{e+1})(\exists y_{e+2}) R(x_1,\dots,x_k,x_{k+1},$$
$$y_1,\dots,y_e,y_{e+1},y_{e+2}),$$

где R - рекурсивное отношение относительно $(\Sigma^\infty)^{k+1}\times(\Sigma^*)^{\ell+2}$.

Будем говорить, что язык $\mathcal{L}$ бесконечных слов принадлежит классу Σ_n (Π_n), если классу Σ_n (Π_n) принадлежит одноместное соотношение $R(x) \equiv (x \in \mathcal{L})$.

ТЕОРЕМА I. Следующие утверждения равносильны: I) язык $\mathcal{L}$ бесконечных слов распознаваем детерминированной машиной Тьюринга; 2) язык $\mathcal{L}$ бесконечных слов выразим с помощью конечного числа булевых операций через языки из класса Σ_2.

ТЕОРЕМА 2. Следующие утверждения равносильны: I) язык $\mathcal{L}$ бесконечных слов распознаваем недетерминированной машиной Тьюринга; 2) язык $\mathcal{L}$ бесконечных слов принадлежит Σ_1^1.

Таким образом, недетерминированные машины Тьюринга имеют значительные преимущества перед детерминированными.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Büchi J.R. On a decision method in restricted second order arithmetic// Proc. International Congress on Logic: Methodology and Philosophy of Science. Stanford: Stanford University Press, 1962, P.1-11.

2. McNaughton R. Testing and generating infinite sequences by a finite automaton // Information and Control, 1966. V.9. P. 521-530.

3. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. М.: Мир, 1972.

ЗАКЛЮЧЕНИЕ

Результаты, опубликованные в данном сборнике, составляют основу для разработки автоматизированных систем тестирования и синтеза программ, которые позволят снизить затраты на создание программного обеспечения ЭВМ. В наши дни во всем мире происходит резкий переход от программирования с использованием примитивной методологии разработки программ к более формализованной технологии программирования, использующей нетривиальным образом всю мощь классической математики и математической логики. Без использования разработанной теории алгоритмов такая технология программирования невозможна.

Результаты получены в рамках НИР "Индуктивные и вероятностные методы в теории программ" и "Синтез и тестирование программ".

Депонированные работы в сборнике не содержатся.

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ ПРОГРАММИРОВАНИЯ

Сборник научных трудов

Рецензенты: И.Страздиньш, к[illegible] физ.-мат.наук, доцент РПИ

Я.Калнинып, канд.физ.-мат.наук, ст.научн.сотр. Института электроники и вычислительной техники АН ЛатвССР;

Э.Икаунискс, канд.физ.-мат.наук, доцент каф. дискретной математики и программирования ЛГУ им. П.Стучки

Редакторы: Р.Грейвалд, Р.Павлова
Технический редактор С.Лининя
Корректор Р.Мурзиня

Подписано к печати 11.11.1988. ЯТ 09476 Ф/б 60x84/16
Бумага №1. 8,8 физ.печ.л. 8,2 усл.печ.л. 6,8 уч.-изд.л.
Тираж 500 экз. Зак. № 1316 Цена 1 р.40к.

Латвийский государственный университет им. П.Стучки
226098 Рига, б. Райниса, 19
Отпечатано на ротапринте, 226050 Рига, ул.Вейденбаума,5
Латвийский государственный университет им. П.Стучки

A B S T R A C T S

INDUCTIVE SYNTACTICAL SYNTHESIS OF PROGRAMS FROM INCOMPLETE SAMPLE COMPUTATIONS

E.B.Kinber

The paper deals with the problem of inductive synthesis of programs from incomplete sample computations. Incomplete sample computations reflect the essential part of the program's behaviour. For example, for a divide-and-conquer algorithm an incomplete sample is the fragment of computation up to the point where the task is divided to subtasks. A formal model convenient for the program synthesis and a synthesis algorithm are developed such that, given a "representative" incomplete sample computation of a program P annotated by variables whose values occur in the sample, the program equivalent to P is synthesized.

BOOLEAN FUNCTIONS OF INFINITE WORDS

M.Alberts

The paper deals with the problem of recognition of infinite languages by one-way deterministic, nondeterministic and probabilistic Turing machines. The number of letters read by a Turing machine is used as a measure of computational complexity. Lower and upper bounds for recognition of different languages by different kinds of Turing machines are obtained. In particular, advantages of probabilistic and nondeterministic computations over deterministic computations are discovered.

PROBABILISTIC ALGORITHMS AND THE STRING CONTINUATION PROBLEM

R.Freivald and M.Chytil

The relation between the problems "given x, decide whether $x \in L$" and "given x, decide whether there is y such that $xy \in L$" is studied. For all the languages considered the first problem can be decided by probabilistic one-wan one-counter machines with arbitrarily high probability $1-\varepsilon$ ($\varepsilon > 0$). Languages $L_2, L_3, \dots$ are constructed such that the second problem for L_k can be decided by probabilistic one-way one-counter machines with probability $k/2k-1$ and not higher.

THE INDUCTIVE SYNTHESIS OF THE GENERAL GRAPHICAL EXPRESSIONS

I.Etmane

A formal language - the so-called general graphical expressions is introduced. This language supports a description of a particular class of graphs. On the other hand, it can be considered as a graphical programming language. A formal example of a graphical expression is defined. A system of the inductive interence rules synthesizing general expressions from sufficiently large formal examples is found and the theorem of its completeness is formulated. The example of the inductive synthesis of the Gauss algorithm from its sample computations within this frame is also given.

ON CAPABILITIES OF ONE-WAY MULTITAPE FINITE AUTOMATA WITH BOUNDED ALTERNATION

D. Geidmanis

Tally languages accepted by multitape one-way finite automata with constant-bounded number of quantifier alternations (in the tree of all possible performances) are proved to be semilinear. Hence every such a language is accepted by a nondeterministic multitape one-way finite automaton as well.

ON THE NUMBER OF LEAVES IN THE ACCEPTATION TREES FOR ALTERNATING ONE-WAY PUSHDOWN AUTOMATA

J.Kaņeps

The minimum number of leaves in the acceptation tree for alternating one-way pushdown automata recognizing a nonregular tally language is found. There is a nonregular tally language recognizable with $c \cdot \log_2 \log_2 \ell$ leaves (c being arbitrary small) where ℓ is the length of the input. No nonregular tally language can be recognized by alternating one-way pushdown automata with $o(\log \log \ell)$ leaves.

RECOGNITION OF ISOMORPHISM OF THE GRAPHS MODELLING INTEGRATED CIRCUITS

P.Ručevskis

An effective algorithm for the recognition of isomorphism of sparse graphs is presented. It is convenient to understand the algorithm as an iterative procedure where finding of an appropriate stabilization of colouring alternates with destabilizing search after another match of classes of vertices. Results of calculations on EC-1060 computer are presented.

RELATIONAL ALGEBRAS IN DATABASES AND GENERALIZED CYLINDRIC ALGEBRAS

J.Cīrulis

It is well-known that cylindric algebras have arised as a result of algebratization of the first-order predicate logic with equality - provided that we only deal with laguages without function symbols. Cylindric set algebras may be used as algebras of relations in relational databases. We note in the paper that it is important in the theory of databases to consider languages which have function symbols, and then one needs to generalize the notion of cylindric algebra in an appropriate way. We propose such a generalization and consider representation problems for generalized cylindric algebras.

ON PLANE TRANSFORMS BASED RECOGNITION OF PLANAR HIERARCHICAL PATTERNS

P.Ķikusts

A functional procedure for the recognition and positioning of patterns in planar scenes containing hierarchically structured objects that may touch or overlap is presented.

THE MINIMUM OF THE QUESTIONS USED IN DECISION TREES

R.Freivald

If a Boolean function has n essential variables then no deterministic decision tree can compute it with less than $\log_2 n$ questions. For nondeterministic decision trees this bound is lowered. A Boolean function of n essential variables is constructed such that a nondeterministic decision tree can compute it with $\frac{1}{2}\log_2 n + o(\log_2 n)$ questions.

RECOGNITION OF LANGUAGES OF INFINITE WORDS BY TURING MACHINES

D.Taimiņa

Languages of infinite words are recognized by deterministic Turing machines iff they are in the Boolean closure of the Kleene-Mostowski class Σ_2^0. Languages of infinite words are recognized by nondeterministic Turing machines iff they are in the Kleene-Mostowski class Σ_1^1.

УДК 519.681.1:714.5

Кинбер Е.Б. Индуктивный синтаксический синтез программ по неполным примерам // Теоретические вопросы программирования. Рига : ЛГУ им.П.Стучки, 1988. С.4-23.

В работе предлагается язык, удобный для индуктивного синтеза большого ряда алгоритмов по неполным примерам вычислений. Для облегчения синтеза примеры снабжаются аннотациями, указывающими, значениями каких переменных являются числа, фигурирующие в примере. Разработан алгоритм синтеза, имеющий полиномиальную (от длины входного примера) сложность.
Библиогр. 5 назв.

УДК 519.95

Албертс М.Я. Булевы функции на бесконечных словах // Теоретические вопросы программирования. Рига : ЛГУ им.П. Стучки, 1988. С.23-33.

В работе рассматривается вычисления на односторонней машине Тьюринга на бесконечных словах в алфавите $\{0,1\}$. Вводится мера сложности - число прочитанных букв $r(n)$ начального отрезка длины n входного слова. Доказаны точные оценки сложности распознавания ряда языков на детерминированных, вероятностных и недетерминированных моделях машины Тьюринга. Результаты демонстрируют преимущества вероятностных и недетерминированных вычислений перед детерминированными.
Библиогр. 3 назв.

УДК 5I9.95

Фрейвалд Р.В., Хитил М.П. Вероятностные алгоритмы и распознаваемость порождаемости слов до языка // Теоретические вопросы программирования. Рига : ЛГУ им.П.Стучки, I988. С.33-50.

Изучается связь между сложностью решения поиска ответов на два типа вопросов для различных языков : I) "принадлежит ли данное слово языку L ?", 2) "может ли данное слово x быть продолжаемо до слова $y \in L$?". Построен ряд языков, для которых ответы на вопросы первого типа могут быть получены со сколь угодно высокой вероятностью вероятностными односторонними счетчиковыми машинами. Сложность решения второй проблемы для этих языков существенно различна. Для $k \geqslant I$ проблема продолжаемости слов до языка L_k разрешима на вероятностных односторонних счетчиковых машинах с вероятностью $k/2k-I$, но не выше. Для языка L_0 эта проблема алгоритмически неразрешима.
Библиогр. 4 назв.

УДК 5I9.68I.I:7I4.5

Этмане И.Э. Индуктивный синтез общих графических выражений // Теоретические вопросы программирования. Рига : ЛГУ им.П.Стучки, I988. С.5I-68.

Введен формальный язык т.н. общих графических выражений, являющихся обобщением языка графических выражений на графы.

Даны примеры использования данного языка для описания алгоритмов типа нахождения решения системы линейных уравнений. Определено понятие формального примера общего графического выражения. Даны правила индуктивного вывода для синтеза общих выражений по их одному достаточно большому примеру. Приводится теорема полноты предложенной системы правил вывода.
Библиогр. 3 назв.

УДК 519.95

Гейдманис Д.Г. О возможностях односторонних многоленточных конечных автоматов с ограничением по альтернированию // Теоретические вопросы программирования. Рига : ЛГУ им. П.Стучки, 1988. С.68-78.

Доказано, что языки в однобуквенном алфавите, принимаемые многоленточными односторонними конечными альтернирующими автоматами с константным ограничением количества перемен кванторов альтернирования (в дереве всех возможных реализаций), являются полулинейными. Поэтому класс таких языков совпадает также с классом языков, принимаемых многоленточными односторонними недетерминированными конечными автоматами.
Библиогр. 9 назв.

УДК 519.95

Канеп Я.Я. О числе листьев принимающего дерева для односторонних альтернирующих магазинных автоматов // Теоретические вопросы программирования. Рига : ЛГУ им.П.Стучки, 1988. С.79-88.

В статье найдено минимальное число листьев принимающего дерева для альтернирующих односторонних магазинных автоматов, распознающих нерегулярный язык в однобуквенном алфавите. Доказывается, что существует нерегулярный язык в алфавите $\{0\}$, распознаваемый для сколь угодно малого c с числом листьев $c\log_2\log_2 l$, где l - длина входного слова. Утверждается, что не существует нерегулярный язык, распознаваемый такими автоматами с числом листьев $o(\log_2\log_2 l)$.
Библиогр. 3 назв.

УДК 519.1

Ручевскис П.Б. Распознавание изоморфизма графов интегральных схем // Теоретические вопросы программирования. Рига : ЛГУ им.П.Стучки, 1988. С.89-100.

Предложена эффективная структура данных для алгоритма распознавания изоморфизма разреженных графов. Алгоритм сочетает рекурсивный перебор с классическим механизмом стабилизации раскраски вершин. Приведены результаты вычислений на ЭВМ ЕС-1060.
Библиогр. 13 назв.

УДК 002.53:681.327

Цирулис Я.П. Алгебры отношений в базах данных и обобщенные цилиндрические алгебры // Теоретические вопросы программирования. Рига : ЛГУ им.П.Стучки, 1988. С.100-107.

Как известно, цилиндрические алгебры возникли в результате алгебраизации логики предикатов первого порядка с равенством - при предположении, что рассматриваются языки без функциональных символов Цилиндрические алгебры множеств могут использоваться в качестве алгебр отношений в реляционных базах данных. В работе отмечается, что в теории баз данных важно рассматривать языки с функциональными символами и что тогда возникает необходимость соответствующим образом обобщить и понятие цилиндрической алгебры. Предлагается такое обобщение и рассматриваются вопросы представимости обобщенных цилиндрических алгебр.
Библиогр. 10 назв.

УДК 681.327

Кикуст П.Б. Распознавание планарных иерархических образов на основе преобразований плоскости // Теоретические воп-

росы программирования. Рига : ЛГУ им.П.Стучки, 1988. С.108-119.

Предложена функциональная процедура распознавания и локализации образов на плоских составных сценах из объектов, имеющих иерархическое строение, возможно соприкасающихся или частично перекрывающих друг друга. Процедура определяет все вхождения заданного образа в изображении и обеспечивает инвариантность распознавания по отношению к допустимым преобразованиям плоскости и к вариациям формы, обусловленным возможностью автономного расположения составных частей образа. Введенный набор функций предполагает наличие вычислительных устройств параллельного действия, однако в демонстрационных целях приведена версия программы для традиционной ЭВМ и пример ее работы.
Библиогр. 17 назв.

УДК 519.95

Фрейвалд Р.В. Минимальное число задаваемых вопросов в деревьях решений // Теоретические вопросы программирования. Рига: ЛГУ им.П.Стучки, 1988. С. 120-122.

Известно, что для вычисления булевой функции от n существенных переменных любое детерминированное дерево решения должно в худшем случае задать не менее чем $\log_2 n$ вопросов. Построена последовательность булевых функций, для вычисления которой недетерминированное дерево решений всегда может обойтись $\frac{1}{2}\log_2 n + O(\log n)$ вопросами.
Библиогр. 1 назв.

УДК 519.95

Тайманов Д.Я. Распознавание языков бесконечных слов на машинах Тьюринга // Теоретические вопросы программирования. Рига: ЛГУ им.П.Стучки, 1988. С.123-125.

Получена характеризация в терминах иерархии Клини-Мостовского языков бесконечных слов, распознаваемых детерминированными и недетерминированными машинами Тьюринга. На детерминированных машинах распознаваемы языки, которые выразимы конечным числом булевых операций над Σ^0_2-множествами. На недетерминированных машинах распознаваемы множества из Σ^1_1 и только они.
Библиогр. 3 назв.

Цена 1 р. 40 к.